US006496860B2

(12) United States Patent
Ludtke et al.

(10) Patent No.: **US 6,496,860 B2**
(45) Date of Patent: ***Dec. 17, 2002**

(54) **MEDIA MANAGER FOR CONTROLLING AUTONOMOUS MEDIA DEVICES WITHIN A NETWORK ENVIRONMENT AND MANAGING THE FLOW AND FORMAT OF DATA BETWEEN THE DEVICES**

(75) Inventors: **Harold Aaron Ludtke**, San Jose, CA (US); **Bruce Fairman**, Woodside, CA (US); **Scott Smyers**, San Jose, CA (US)

(73) Assignees: **Sony Corporation**, Tokyo (JP); **Sony Electronics, Inc.**, Park Ridge, NJ (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/801,429**

(22) Filed: **Mar. 7, 2001**

(65) **Prior Publication Data**

US 2001/0018718 A1 Aug. 30, 2001

Related U.S. Application Data

(63) Continuation of application No. 09/075,047, filed on May 8, 1998, now Pat. No. 6,233,611.

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **709/223**; 709/224
(58) **Field of Search** 709/223, 220, 709/221, 222, 201, 224, 200, 203, 217, 218, 219

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,562,535 A	12/1985	Vincent et al.	364/200
4,633,392 A	12/1986	Vincent et al.	364/200

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

DE	38 12 607 A1	10/1988
EP	0 499 394 A1	8/1992
EP	0 588 046 A1	3/1994
EP	0 745 929 A1	12/1996
GB	2 203 869 A1	10/1988

OTHER PUBLICATIONS

IEEE, "1394–1995 Standard for a High Performance Serial Bus," 1995, pp. 1–384, IEEE.

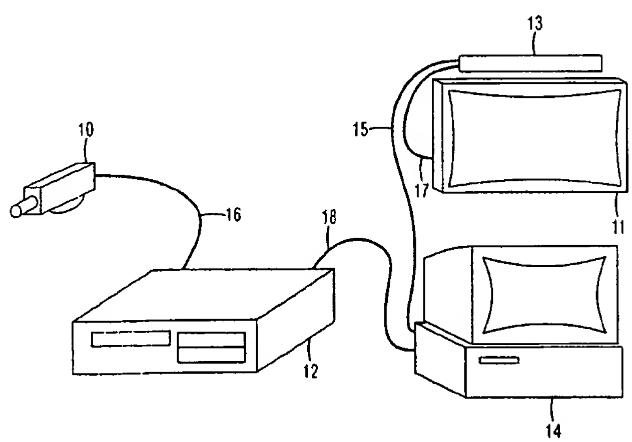
(List continued on next page.)

Primary Examiner—Moustafa M. Meky
(74) *Attorney, Agent, or Firm*—Haverstock & Owens LLP

(57) **ABSTRACT**

A media manager provides data flow management and other services for client applications on devices coupled together within a network. Preferably, these devices are coupled together within an IEEE 1394–1995 serial bus network. A device control module is generated for each available device for providing an abstraction for all of the capabilities and requirements of the device including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfers between the devices on the network. Through an interface, a user accesses the media manager and enters functions which are to be completed using the devices coupled together on the network. If the appropriate devices are available, the media manager controls and manages the completion of the requested task. If the appropriate devices are not available, but the required subdevices are available in multiple devices, the media manager forms a virtual device from subdevices in multiple devices in order to complete the requested task. Once the appropriate devices and subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from one format into another format. If necessary, the media manager will also control the format conversion during the data transfer operation. The media manager also provides network enumeration and registry searching capabilities for client applications to find available services, physical devices and virtual devices.

43 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

5,289,460	A		2/1994	Drake, Jr. et al.	370/17
5,394,556	A		2/1995	Oprescu	395/800
5,457,446	A		10/1995	Yamamoto	340/825.24
5,574,965	A		11/1996	Welmer	455/3.2
5,606,664	A	*	2/1997	Brown et al.	709/105
5,621,662	A	*	4/1997	Humpheries et al.	364/550
5,621,901	A		4/1997	Morriss et al.	395/306
5,715,475	A		2/1998	Munson et al.	395/830
5,724,272	A	*	3/1998	Mitchell et al.	364/579
5,724,517	A	*	3/1998	Cook et al.	709/227
5,793,366	A	*	8/1998	Mano et al.	364/329
5,809,249	A		9/1998	Julyan	395/200.53
5,815,082	A		9/1998	Welmer	340/825.07
5,815,678	A		9/1998	Hoffman et al.	395/309
5,920,479	A	*	7/1999	Sojood et al.	364/191
5,940,387	A	*	8/1999	Humpleman	370/352
5,963,726	A	*	10/1999	Rust et al.	395/500
6,233,611	B1	*	5/2001	Ludtke et al.	709/223

OTHER PUBLICATIONS

R.H.J. Bloks, "The IEEE–1394 High Speed Serial Bus," Philips J. Res. 50, pp. 209–216, 1996.

Walter Y. Chen, "Emerging Home Digital Networking Needs," Sep. 11–12, 1997, IEEE Fourth International Workshop on Community Networking Proceedings, pp. 7–12.

Jerrold A. Friesen, "Dave: A Plug–and–Play Model for Distributed Multimedia Application Development," Proceedings ACM Multimedia 94, pp 22–28, IEEE Parallel & Distributed Technology.

Julia L. Heeter, "Asynchronous Transfer Mode," Dec. 12, 1995., pp 1–11, http://www.gl.umbc.edu/~jheetel/atmpaper.html.

Gary Hoffman et al., "IEEE 1394: A Ubiquitous Bus," Digest of Papers of the Computer Society Computer Conference (Spring) Compcom, U.S., Los Alamitos, IEEE Comp. Soc. Press, vol. CONF. 40, Mar. 5, 1995, pp. 334–338.

Michael Teener et al., "A Bus on a Diet—The Serial Bus Alternative An Introduction to the P1394 High Performance Serial Bus," Feb. 24, 1992, pp. 316–321, IEEE.

Ingrid J. Wickelgren, "The Facts About FireWire," IEEE Spectrum, vol. 34, No. 4, Apr. 1997, pp. 19–25.

Maury Wright, "USB and IEEE 1394: Pretenders, Contenders, or Locks for Ubiquitous Desktop Deployment?" EDN Electrical Design News, U.S. Cahners Publishing Co., Newton Mass., vol. 41, No. 9, Apr. 25, 1996, pp. 79–80, 82, 84,86.

* cited by examiner

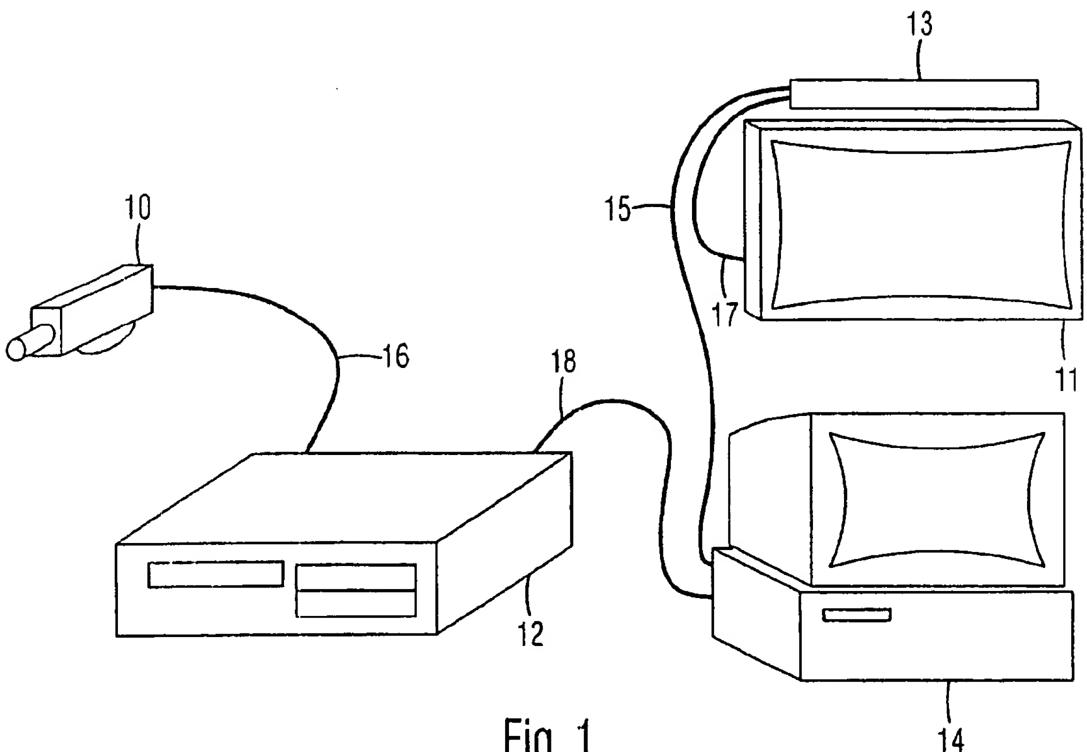


Fig. 1

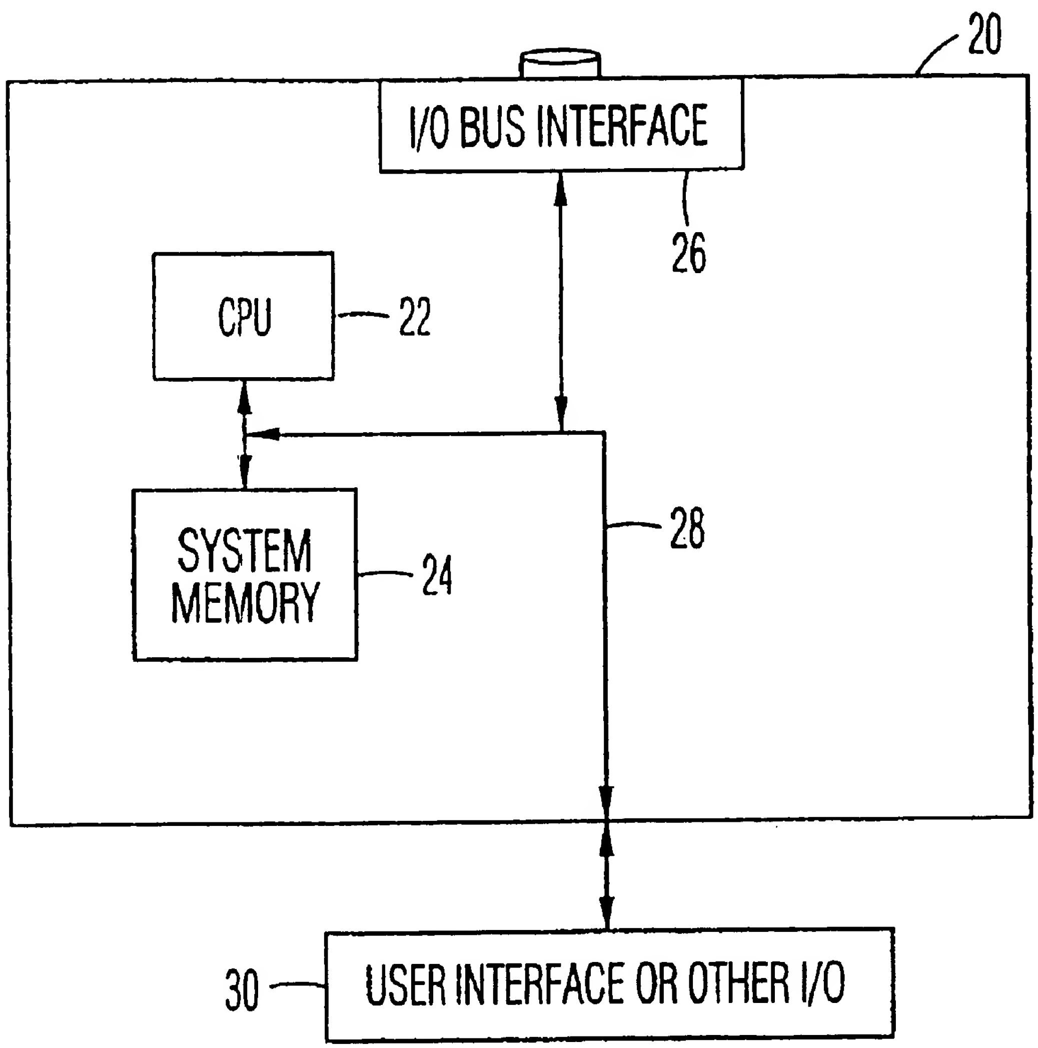

20
I/O BUS INTERFACE
26
CPU
22
SYSTEM MEMORY
24
28
30
USER INTERFACE OR OTHER I/O

Fig. 2

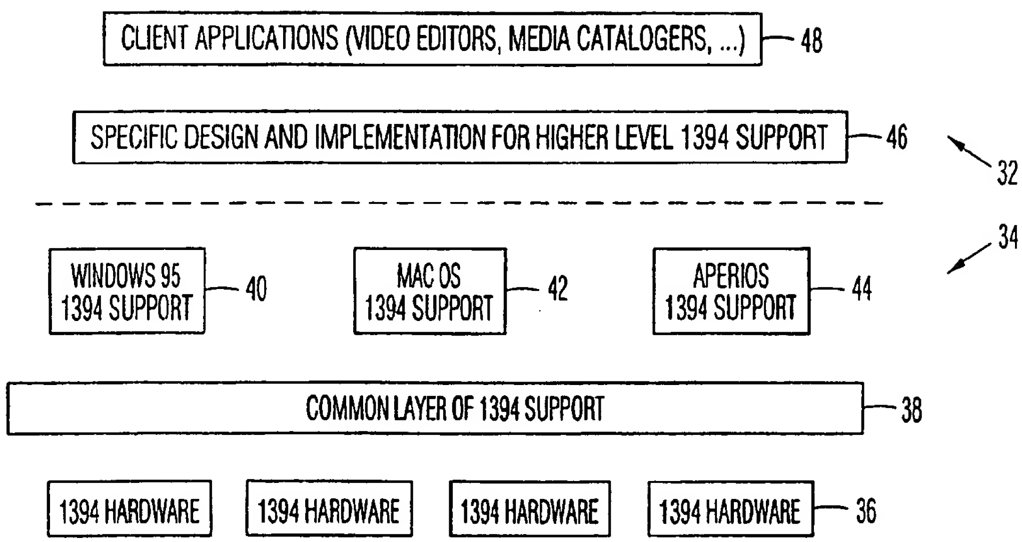

CLIENT APPLICATIONS (VIDEO EDITORS, MEDIA CATALOGERS, ...)
48
SPECIFIC DESIGN AND IMPLEMENTATION FOR HIGHER LEVEL 1394 SUPPORT
46
32
34
WINDOWS 95 1394 SUPPORT
40
MAC OS 1394 SUPPORT
42
APERIOS 1394 SUPPORT
44
COMMON LAYER OF 1394 SUPPORT
38
1394 HARDWARE
1394 HARDWARE
1394 HARDWARE
1394 HARDWARE
36

Fig. 3

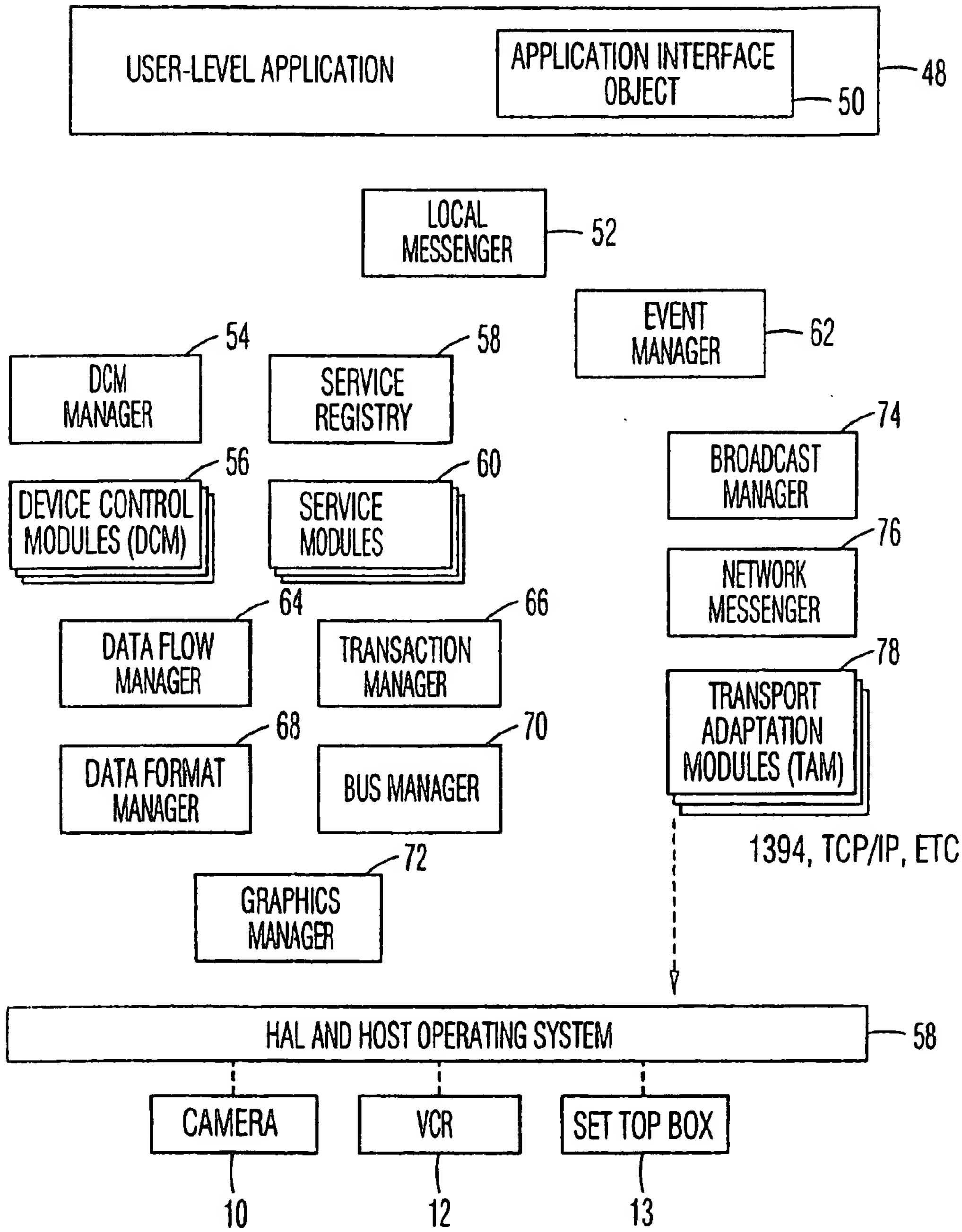

USER-LEVEL APPLICATION
APPLICATION INTERFACE OBJECT
50
48
LOCAL MESSENGER
52
EVENT MANAGER
62
DCM MANAGER
54
SERVICE REGISTRY
58
BROADCAST MANAGER
74
DEVICE CONTROL MODULES (DCM)
56
SERVICE MODULES
60
NETWORK MESSENGER
76
DATA FLOW MANAGER
64
TRANSACTION MANAGER
66
TRANSPORT ADAPTATION MODULES (TAM)
78
DATA FORMAT MANAGER
68
BUS MANAGER
70
1394, TCP/IP, ETC.
GRAPHICS MANAGER
72
HAL AND HOST OPERATING SYSTEM
58
CAMERA
VCR
SET TOP BOX
10
12
13

Fig. 4

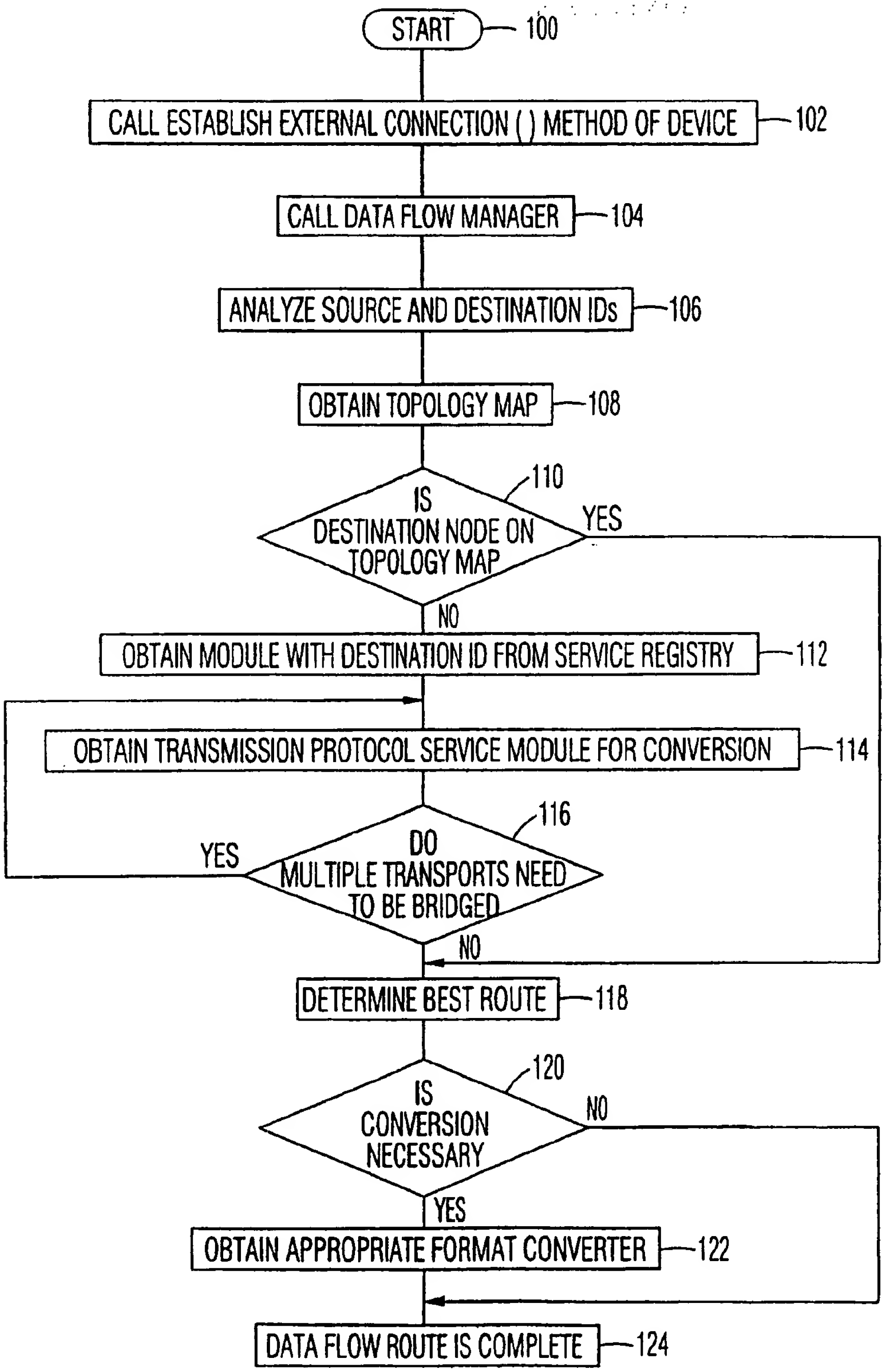

START
100
CALL ESTABLISH EXTERNAL CONNECTION ( ) METHOD OF DEVICE
102
CALL DATA FLOW MANAGER
104
ANALYZE SOURCE AND DESTINATION IDs
106
OBTAIN TOPOLOGY MAP
108
IS DESTINATION NODE ON TOPOLOGY MAP
110
YES
NO
OBTAIN MODULE WITH DESTINATION ID FROM SERVICE REGISTRY
112
OBTAIN TRANSMISSION PROTOCOL SERVICE MODULE FOR CONVERSION
114
DO MULTIPLE TRANSPORTS NEED TO BE BRIDGED
116
YES
NO
DETERMINE BEST ROUTE
118
IS CONVERSION NECESSARY
120
NO
YES
OBTAIN APPROPRIATE FORMAT CONVERTER
122
DATA FLOW ROUTE IS COMPLETE
124

Fig. 5

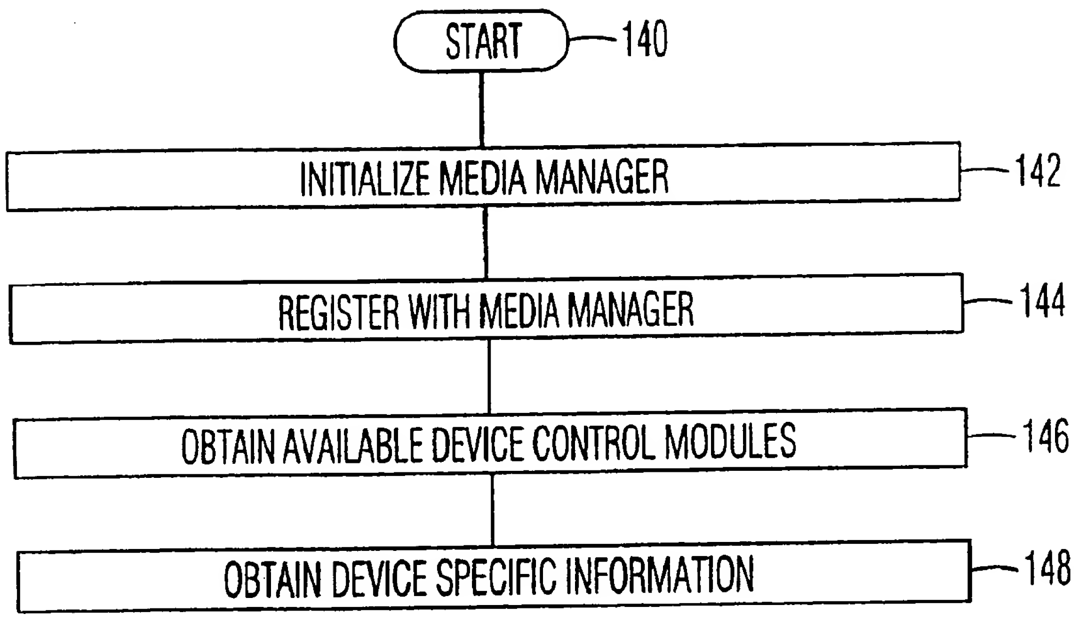

START
140
INITIALIZE MEDIA MANAGER
142
REGISTER WITH MEDIA MANAGER
144
OBTAIN AVAILABLE DEVICE CONTROL MODULES
146
OBTAIN DEVICE SPECIFIC INFORMATION
148

Fig. 6

MEDIA MANAGER FOR CONTROLLING AUTONOMOUS MEDIA DEVICES WITHIN A NETWORK ENVIRONMENT AND MANAGING THE FLOW AND FORMAT OF DATA BETWEEN THE DEVICES

This application is a continuation of U.S. patent application Ser. No. 09/075,047 filed on May 8, 1998 now U.S. Pat. 6,233,611.

FIELD OF THE INVENTION

The present invention relates to the field of managing applications and devices within a network environment. More particularly, the present invention relates to the field of managing the operation of and the communication between devices within a network environment.

BACKGROUND OF THE INVENTION

The IEEE 1394–1995 standard, "1394–1995 Standard For A High Performance Serial Bus," is an international standard for implementing an inexpensive high-speed serial bus architecture which supports both asynchronous and isochronous format data transfers. Isochronous data transfers are real-time transfers which take place such that the time intervals between significant instances have the same duration at both the transmitting and receiving applications. Each packet of data transferred isochronously is transferred in its own time period. An example of an ideal application for the transfer of data isochronously would be from a video recorder to a television set. The video recorder records images and sounds and saves the data in discrete chunks or packets. The video recorder then transfers each packet, representing the image and sound recorded over a limited time period, during that time period, for display by the television set. The IEEE 1394–1995 standard bus architecture provides multiple channels for isochronous data transfer between applications. A six bit channel number is broadcast with the data to ensure reception by the appropriate application. This allows multiple applications to simultaneously transmit isochronous data across the bus structure. Asynchronous transfers are traditional data transfer operations which take place as soon as possible and transfer an amount of data from a source to a destination.

The IEEE 1394–1995 standard provides a high-speed serial bus for interconnecting digital devices thereby providing a universal I/O connection. The IEEE 1394–1995 standard defines a digital interface for the applications thereby eliminating the need for an application to convert digital data to analog data before it is transmitted across the bus. Correspondingly, a receiving application will receive digital data from the bus, not analog data, and will therefore not be required to convert analog data to digital data. The cable required by the IEEE 1394–1995 standard is very thin in size compared to other bulkier cables used to connect such devices. Devices can be added and removed from an IEEE 1394–1995 bus while the bus is active. If a device is so added or removed the bus will then automatically reconfigure itself for transmitting data between the then existing nodes. A node is considered a logical entity with a unique address on the bus structure. Each node provides an identification ROM, a standardized set of control registers and its own address space.

Media devices are being equipped with network interfaces allowing them to become part of a network such as the IEEE 1394–1995 serial bus network. In a home audio/video network incorporating such autonomous media devices it is

possible that one or more such devices will be coupled together in a network with a personal computer, settop box or other device including a microprocessor. Currently, there is a lack of available interfaces and control applications which will efficiently manage interaction and operation of the autonomous devices within such a network configuration. What is needed is an interface which allows a controlling device within a network configuration to efficiently control communications between the devices and the operation of the devices within the network. What is further needed is an interface which allows a controlling device within a network configuration to maximize the availability of devices within a network for completion of tasks and operations.

SUMMARY OF THE INVENTION

A media manager provides data flow management and other services for client applications on devices coupled together within a network. Preferably, these devices are coupled together within an IEEE 1394–1995 serial bus network. A device control module is generated for each available device for providing an abstraction for all of the capabilities and requirements of the device including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfers between the devices on the network. Through an interface, a user accesses the media manager and enters functions which are to be completed using the devices coupled together on the network. If the appropriate devices are available, the media manager controls and manages the completion of the requested task. If the appropriate devices are not available, but the required subdevices are available in multiple devices, the media manager forms a virtual device from subdevices in multiple devices in order to complete the requested task. Once the appropriate devices and subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from one format into another format. If necessary, the media manager will also control the format conversion during the data transfer operation. The media manager also provides network enumeration and registry searching capabilities for client applications to find available services, physical devices and virtual devices.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an IEEE 1394–1995 serial bus network including a video camera, a video cassette recorder, a computer, a television and settop box.

FIG. 2 illustrates a block diagram of a hardware system resident in each device implementing the media manager of the present invention.

FIG. 3 illustrates a block diagram of the architecture of the media manager platform of the present invention.

FIG. 4 illustrates a detailed block diagram of the architecture of the media manager platform of the present invention.

FIG. 5 illustrates a flow diagram of the steps involved in setting up and data transfer between two devices using the media manager of the present invention.

FIG. 6 illustrates a flow diagram followed by a client application during startup.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The media manager of the present invention provides data flow management and other services for physical devices

within a network. A physical device is a product sold as a separate component by a vendor. Examples of physical devices include televisions, video cassette recorders, personal computers, video cameras, CD-Rom players and the like. Many other examples are also well known and commercially available. Each physical device includes a number of subdevices. For example, a typical commercially available video camera includes multiple subdevices, implementing different functionalities, such as the camera and the video player.

Preferably, these physical devices are coupled together within an IEEE 1394–1995 serial bus network. A device control module (DCM) is generated for each available device and subdevice. Each DCM provides an abstraction for all of the capabilities and requirements of each physical device, including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfer operations between the physical devices on the network, including converting the data into a different format during the data transfer operation.

Through an interface, a user accesses the media manager and enters functions and tasks which are to be completed using the physical devices within the network. If the appropriate physical devices are available and are not otherwise being used, the media manager controls and manages the completion of the requested task. If the appropriate physical devices are not available, the media manager attempts to create a virtual device from available subdevices or components within devices in order to complete the requested task. Once the appropriate physical devices and/or subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from the format of the source device to the format of the receiving device. If a conversion is necessary, the media manager also controls that operation while controlling the data transfer operation.

FIG. 1 illustrates an exemplary system including a video camera **10**, a video cassette recorder **12**, a computer **14**, a settop box **13** and a television **11** connected together by the IEEE 1394–1995 cables **15**, **16** and **18**. The IEEE 1394–1995 cable **16** couples the video camera **10** to the video cassette recorder **12**, allowing the video camera **10** to send data to the video cassette recorder **12** for recording. The IEEE 1394–1995 cable **18** couples the video cassette recorder **12** to the computer **14**, allowing the video cassette recorder **12** to send data to the computer **14** for display. The IEEE 1394–1995 cable **15** couples the settop box **13** to the computer **14**. The settop box **13** is also coupled to the television **11** by the cable **17**.

This configuration illustrated in FIG. 1 is exemplary only. It should be apparent that an audio/video network could include many different combinations of physical components. The physical devices within such an IEEE 1394–1995 network are autonomous devices, meaning that in an IEEE 1394–1995 network, as the one illustrated in FIG. 1, in which a computer is one of the devices, there is not a true "master-slave" relationship between the computer and the other devices. In many IEEE 1394–4995 network configurations, a computer may not even be present. Even in such configurations, the devices within the network are fully capable of interacting with each other on a peer basis.

A block diagram of a hardware system resident in the managing device for implementing the media manager of the present invention is illustrated in FIG. 2. In the hardware system illustrated in FIG. 2, a printed circuit board **20** is coupled to a user interface **30**. The printed circuit board **20** includes a central processing unit (CPU) **22** coupled to system memory **24** and to an I/O bus interface **26** by the system bus **28**. The use of the term 'CPU' is not intended to imply that such a system must be a general purpose computing circuit. Rather, this circuit could be implemented with a general purpose controller or special purpose circuit. The user interface **30** is also coupled to the system bus **28**. The user interface **30** is subsystem specific, but preferably includes at least an infrared remote control device and a display. Alternatively, the user interface **30** also includes other I/O devices for communicating with a user of the subsystem.

In the preferred embodiment of the present invention, the media manager is included within a device such as a television or a computer with display, in order to facilitate smooth interaction with the user. However, it should be apparent that the media manager of the present invention can also be implemented on any other capable device which includes the components necessary for providing an interface to the user. In order to implement the media manager of the present invention, each component in which it is implemented will include a hardware system such as the system illustrated in FIG. 2. The CPU **22** within such a device is used to execute the application program instructions. The media manager of the present invention is then used to manage communications and operations within the network. The user accesses the media manager through the interface provided at the controlling device. Through this interface the user can monitor the operation and status of the network and the devices within the network. The user can also control the devices and request tasks to be completed through this interface. An example of these tasks include playing a recorded tape on the VCR **12** and displaying the output from the VCR **12** on the television **11**. The media manager of the present invention also manages data transfer operations and tasks requested at the individual devices.

A block diagram of the architecture of the media manager platform of the present invention is illustrated in FIG. 3. This architecture is divided into a so-called upper portion 32 and a lower portion 34. The lower portion **34** preferably includes the IEEE 1394–1995 bus interface and functionality support across the most common commercial operating systems currently available. The upper portion **32** includes components which bring together the underlying IEEE 1394–1995 bus support and add in a number of features and enhancements which are provided to the client applications and therefore to the user. The upper portion 32 includes the block **46**, which provides specific design and implementation for higher level IEEE 1394–1995 bus support, and the block **48**, which includes and provides interfaces to the various client applications. The lower portion **34** includes the blocks **40**, **42** and **44** which provide support for the most common operating systems, including Windows 95®), Macintosh® and Aperios™ operating systems, respectively. Support for any general operating system, such as OS9, is also provided. The lower portion **34** also includes the block **38**, which provides the common layer of IEEE 1394–1995 support, and the blocks **36** which provide the actual physical IEEE 1394–1995 bus interfaces to other devices coupled to the controlling device.

The media manager platform provides an open and flexible architecture in order to efficiently integrate personal computers and other autonomous devices into a network configuration and effectively manage the necessary data transfer operations between those devices. The lower portion **34** of the architecture has been designed to support the

underlying technology at the lowest levels, which allows the higher levels to support more general modules and functional descriptions.

A more detailed block diagram of the architecture of the media manager platform of the present invention is illustrated in FIG. 4. The multimedia or user-level application **48** sits at the top of the architecture and makes use of the services provided by the media manager. The multimedia application **48** is an application or other client of the media manager platform of the present invention. The architectural components within the media manager manage the protocol specifics and export a simpler programming interface up to the application **48**. Issues such as timing, buffer management, bus management and communication protocol are hidden behind these simple functional interfaces. The application **48** also has access to the lower layers of the architecture and will of course be able to communicate directly with the hardware adaptation layer (HAL) and the host operating system **58**. The host operating system is coupled to the other devices within the network, such as the camera **10**, the VCR **12** and the settop box **13**. For illustration purposes, in this configuration the media manager of the present invention is implemented on the computer system **14** of FIG. **1**.

The application interface object **50** serves as a proxy for the client application **48** within the media manager environment. An applications programming interface is provided to allow the client applications **48** to access the basic services of the media manager. Access to more detailed or specific functionality provided by certain programming interfaces is also provided through the application interface object **50** which provides the application **48** access to the local messenger **52**.

The local messenger **52** is one component of the messaging system integrated into the media manager. Preferably, this messaging system is the AV Messenger system. The local messenger **52** is the central hub of communications between all objects on a given node, when those objects exist in separate execution spaces. Essentially, the local messenger **52** is the inter-application communication model which is provided by the host operating system **58**. The local messenger **52** is the bottleneck through which all messages between software modules pass. To achieve scriptability, the local messenger **52** logs all messages as they pass through, keeping an internal database of all messages and their relevant data including address of destination, parameters, address of response and optionally, a time stamp for time-based scripting.

The service registry **59** includes a reference to all addressable entities within the media manager **71**. This registry includes a reference for each device control module (DCM) **56**, the DCM Manager **54**, the data flow manager **64**, the transaction manager **66**, the data format manager **68**, the bus manager **70** and the graphics manager **72**. The service registry **59** also contains any number of service modules **60**, as will be described below. The service registry **59** also contains a service registry database including references for all of the objects that are local to its node and at specific times references to remote objects as well. Each entry in the database refers to an addressable module and includes attached attributes, some of which are common to all entries and others which are specific to a type of module. Common attributes include such things as the module name and the local ID. Module-specific attributes will vary by the type of module. Once the entry exists in the service registry database, any number of attributes can be added to an entry. When a client application searches the database, the application specifies a set of attributes to match and the service registry **59** searches the database, finding and returning all entries which match the specified criteria. If multiple candidates are found during this search, the service registry **59** will provide a list reference to the client application **48**. The client application can then examine each of the candidate items in the list, to determine the items of interest.

A client application **48** may have multiple outstanding search lists, each representing the results of a different search criteria. When the client application **48** needs to update a search list, in response to an event such as a bus reset in which different devices may be available, the application **48** passes the list reference back to the service registry **59** when making a search call. This allows the service registry **59** to update the existing list object, rather than disposing of it and reallocating a new one.

The service modules **60** are modules which can be called to perform some set of services. The service modules **60** perform a variety of services for client applications, including such services as data format, transport and control protocol translation.

The DCM manager **54** is responsible for handling the group of DCMs **56** on its local node or for the devices within the controlling device's network. These responsibilities include the tasks of discovering, instantiating and disposing of all possible DCM candidates that are available to a given system. In addition, the DCM manager **54** communicates with other DCM managers on remote nodes, if any, to arbitrate for network-wide device and subdevice resource allocation and management.

The DCM manager **54** works with the underlying operating system services to get a raw list of available device node handles. The DCM manager **54** also provides an application programming interface for the client application **48** to discover what subdevices, represented by DCMs **56**, or other services are available within the devices on the network. A DCM **56** represents a device or subdevice available for allocation by the DCM manager **54**. A DCM **56** can represent a physical device or a virtual device formed of subparts from different physical devices. The other available services are represented by virtual DCMs **56**, which will be discussed below. The available DCMs will be dynamic, depending on the available physical devices on the IEEE 1394–1995 serial bus.

For each node, the DCM manager **54** does enough work to determine that it should create a DCM **56**. This is done for all media-related devices which will be managed under the umbrella of the media manager of the present invention. For each media-related entity, the DCM manager **54** creates a generic DCM **56**. Each DCM **56** then has the responsibility to make itself more device-specific, as will be described below.

Device-specific DCMs provided from manufacturers can also be added into the DCMs **56**. Device-specific DCMs can come from a variety of sources including an embedded read-only memory (ROM) within the device or some other storage mechanism such as the header of a disc or tape. The device-specific DCM may also be downloaded from an internet site or via a direct modem connection, if such capabilities are accessible to the media manager, or supplied from a disk or other storage medium. These alternatives are discussed in detail in the U.S. patent application Ser. No. 09/092,703, filed on Jun. 4, 1998 and entitled "A METHOD AND APPARATUS FOR INCLUDING SELF-DESCRIBING INFORMATION WITHIN DEVICES," which is hereby incorporated by reference.

The DCM manager **54** is responsible for adding and disposing DCMs **56** at the appropriate times, and notifying clients that DCMs **56** have been added or removed. The DCM manager **54** is also responsible for coordinating complex services among multiple DCMs **56**. These complex services, such as command queuing of complex operations, require the DCM manager **54** to coordinate with multiple DCMs **56** to carry out these operations.

The DCMs **56** each provide a device modeling and control protocol abstraction service by exporting a standardized interface for device control up to the client application **48**. The programming interface for device control provided by the DCMs **56** is divided into common A/V control and device-specific A/V control. The common A/V control commands are common to virtually all devices having audio-visual capabilities. The basic transport control functionality such as Play, Stop, Fast Forward and Rewind commands are included here. The device-specific A/V control commands include features common to a given category of A/V devices, such as the Record command for devices with recording capability, and features that are specific to a certain model or group of devices. The information for device-specific functionality can either be built into a device-specific DCM which is embedded in the device itself, using the self-describing data structures mentioned previously, or it can be downloaded as a software upgrade from the internet.

The media manager of the present invention employs protocol abstraction which means that the programming interface between the modules and the application is the same, regardless of the kind of device and the controlling protocol being used. Accordingly, the application will use the same source code and message to cause an IEEE 1394–1995 VCR to record as it would use to cause a Video System Control Architecture (VISCA) VCR to record. This is true for the common A/V control commands and the category-specific control commands; features that are truly specific to a particular device will have a unique programming interface.

The DCM **56** is the mechanism by which self-describing data from a device is downloaded and presented to the user. This requires the DCM **56** to analyze the self-describing information, by downloading and integrating modules and managing the presentation of this information to the user through a host application. This allows a user to configure and control both the well-known and device-specific functionality of devices on the network through the media manager interface. The preferred presentation of user interface data and access to custom functions of the device is described in the U.S. Provisional Patent Application Ser. No. 60/054,199, filed on Jul. 30, 1997 and entitled "METHOD FOR DESCRIBING THE HUMAN INTERFACE FEATURES AND FUNCTIONALITY OF AV/C-BASED DEVICES," which is hereby incorporated by reference.

Together, the DCM manager **54** and the DCMs **56** perform command queuing of AV commands to be executed, allowing the DCM **56** to deal with all device peculiarities, such as the need to perform a pre-roll to account for mechanical latency of a device. The DCM manager **54** and the DCMs **56** in coordination with other parts of the media manager also provide the ability to specify device control actions that are taken at specific times and as the result of certain conditions.

The DCMs **56** make up a large part of the overall architecture of the media manager of the present invention. The DCM **56** provides an abstraction for all of the various pieces of technology that make up an audio-visual device, such as the control protocol, physical connections and connection capabilities. A DCM **56** can also be created which does not represent a physical device, but represents a virtual device comprising a collection of functions or services that carry out a particular AV operation.

Physical devices and subdevices are individually accessible pieces of hardware. The media manager of the present invention uses accessible subdevices to support virtual devices to add enhanced capability to a network of devices. The virtual devices are logical entities which are combined from pieces of a variety of available components. Preferably, the virtual devices are created automatically when necessary to complete a requested task. Alternatively, the virtual devices are created dynamically by requesting the service from the DCM manager **54**.

An AV action is a pre-defined action or activity such as "watch television" or "record a movie," or any set of user-defined actions involving the manipulation of devices by using the DCMs **56**. The actions can be recorded for later use by a user. An A/V action applications programming interface is a way of modeling common actions that are performed with devices in an AV network, such as viewing a recorded show, viewing a broadcast show, duplicating a tape and listening to a compact disk. For example, if a VCR is located in an upstairs bedroom within a user's house and is currently receiving a broadcast through the tuner and displaying it on a television in the bedroom, the transport mechanism within the VCR is not being used. If the user then desires to view a recorded show on the television downstairs, the media manager of the present invention will allow the user to place the video tape in the transport of the VCR in the bedroom and watch the recorded show from the tape on the downstairs television. The data representing the recorded show is transmitted from the upstairs VCR over the IEEE 1394–1995 serial bus network to the downstairs television. This data transfer operation is controlled by the media manager in the controlling device. Similar functions and virtual devices are achieved with tuners, demuxers, transports, amplifiers, processors and other components and subdevices. Accordingly, the user can maximize the functionality and capability of the devices within the network using the media manager to control their operation.

The DCM manager **54** keeps track not only of what physical devices and subdevices are being used, but also what virtual devices can be created from components and subcomponents that are currently available. The DCM manager **54** does this for all of its locally managed devices and for software services available on the host platform on which it is executing. The DCM manager **54** also provides the programming interfaces for a client application **48** to inquire about the virtual devices that can be created based on the resources available on the network, as well as what AV actions can be currently performed. The DCM manager **54** also ensures that virtual devices get added to the service registry **59** at the appropriate times.

The applications programming interface provided for the DCMs **56** is designed to allow the client applications **48** to discover what features and capabilities are available within the devices in the network and then work with those devices as necessary. This programming interface includes device control, device management, connection management and management of the self-describing device implementation. Each of the DCMs **56** have the responsibility to convert from a generic DCM to a device or protocol specific DCM by determining the type of device it is managing. This requires that the DCM examine the self-describing device data from

the device, if any is present, and analyze any other information that may be available. The DCMs 56 also have the responsibility to provide access to the self-describing device information data for the device being managed, including the device image, a product name string and functional descriptors to other devices and components. The DCM 56 is further responsible for providing a consistent interface for device control, including the complex services such as command queuing. Carrying out these commands requires coordination with the host operating system 58 for device control protocol usage, including packaging, sending, processing protocol-specific commands and responses via the protocol driver and other operating system provided support mechanisms.

Each DCM 56 also monitors the device it is controlling and provides extended notification support to the necessary components and applications. All normal events generated by the device go through the DCM 56 for the appropriate device, to the event manager 62 and to all interested client applications 48. In addition to supporting the AV/C device notification events, many situations may not be supported explicitly in either the AV/C protocol, in a given device or other control protocol. Depending on the capabilities of the device and its control and communications protocols, it is possible to provide extended support for such events which do not trigger actual event messages. The DCM 56 watches the device for this kind of activity and posts an event to the event manager 62.

Each DCM 56 is also responsible for managing themselves in terms of the outside entities which are making use of the data from the device under their control and the entities that are controlling them. This includes support for resource sharing and resource queuing. Resource queuing allows an entity to reserve a busy DCM for its use, as soon as the DCM 56 is available. As soon as the DCM 56 is available, it will then notify the entity.

The DCMs 56 also preferably maintain a presence during environmental changes allowing the DCM and clients to support both on-line and off-line states. This allows the DCMs 56 to quickly re-establish the services of a device once it comes on-line again.

Within the media manager of the present invention, the DCMs 56 are responsible for controlling the available devices and subdevices. The DCMs 56 provide access to the capabilities of the device, both general and device-specific. In an alternate embodiment of the present invention, each device, as part of the self-describing data, has an embedded DCM, ensuring that the software is always available regardless of where the device is taken. In a further alternate embodiment, the DCM for a specific device is obtained from the device manufacturer or a third party over the internet or provided on a media device, such as a floppy disk. In each of the above embodiments, the DCM 56, once downloaded can be stored in a variety of locations. Preferably, the DCM 56 is stored on the device it controls. However, the DCM 56 can also be stored in any appropriate location. In an alternate embodiment of the present invention, the DCM 56 is written in common byte or script code format, such as Java or Java Script, supported by the host platform. The DCM 56 is then uploaded to the host device and executed there.

The event manager 62 broadcasts all event notifications within the network to all interested parties. The event manager 62 acts as a central location for all modules within its node to register for notification when events occur in that node. The event manager 62 maintains an event notification list data structure which is a list of the defined event types and the destination identifiers of all devices which have registered for notification of each type of event. The devices each register with the event manager 62 for each event type that is of interest to them, providing their client identifier and a token value to be passed back to them when the event message is broadcast. An event is an actual occurrence of some action and a message from a device which is addressed to multiple destinations.

The event manager 62 does not usually generate events, but accepts and broadcasts events posted by other components with the media manager. When broadcasting events to client applications 48 in remote nodes, the event manager 62 makes use of the broadcast manager 74. The event manager 62 handles the interaction with the user and informs the appropriate devices accordingly. The event manager 62 informs the-DCMs 56 of what user input is occurring through the interface at the control software level, so that the DCMs 56 can handle their physical devices appropriately. A DCM 56 controlling its device from a remote location will need to receive messages indicating what the user is doing and will need to send appropriate messages to its device. The event manager 62 supports the execution of remote DCMs 56 by means of the messaging system and well-defined event messages. The well-defined event messages include device administration, such as a new_device message generated when a device is added to the network, and user interaction. The user interaction messages support the preferred graphical user interface model as described in the U.S. Provisional Patent Application Ser. No. 60/054,199, filed on Jul. 30, 1997 and entitled "METHOD FOR DESCRIBING THE HUMAN INTERFACE FEATURES AND FUNCTIONALITY OF AV/C-BASED DEVICES," which is hereby incorporated by reference. In addition to well-defined event messages, any two DCMs or software modules can also define custom or private messages.

The graphics manager 72 manages the embedding of remote device controls into a controlling application and supports the remote presentation of self-describing data information by the DCMs 56. The graphics manager 72 provides a programming interface that allows the DCMs 56 to arbitrate for screen space and to work within a shared graphics environment. This allows the specific capabilities of a device to be presented to and accessed by the user through the interface of the controlling software.

The data format manager 68 manages the format of the data flowing between devices. This includes the ability to plug into the resident media manager for data format conversion as part of the buffer management and data format process. Most of the data format conversions are done transparently on behalf of the client application, based on knowledge of the source and destination of the data. Other data transformations require the client application 48 to set up a format conversion process. Preferably, the format conversion is performed in-line while the data is being transmitted. Alternatively, the format conversion is performed as either a pre or post processing task to transmission of the data. The data format conversion services available on a given platform are stored in the service registry 59. In addition to using the registry to find services, the data format manager 68 is responsible for instantiating the service modules and registering them with the service registry 59.

The data flow manager 64 works with the bus manager 70 to provide services to assist with routing data from source to destination, which may include many nodes in between. In the event that the source and destination device involve different data types, or are separated by a barrier, the data flow manager 64 will work with the data format manager 68

and the service registry 59 to handle automatic or requested data translation services as well. During the transfer of isochronous data, the data flow manager 64 provides buffer allocation and management services. Buffer management includes the provision for a consistent notification mechanism to inform the client application when data is available for processing. While isochronous data is flowing into the client application 48, various memory buffers are filled with the data. The data flow manager 64 informs the client application 48 when each buffer is filled so that it can handle the data acquisition process from the buffer. In addition, buffer management is simplified for client applications by having the appropriate service modules partition memory in a manner that is optimized for the data being captured. This includes separating the allocated memory into scan line or frame-sized segments for a stream of video data or the optimum segment sizes for raw audio and MIDI data.

A flow diagram of the steps involved in setting up a data transfer between two devices using the media manager of the present invention is illustrated in FIG. 5. The method starts at the block 100. When a client application 48 desires to establish a connection between two devices for a transfer of data, the application calls the EstablishExternalConnection( ) method of one of the two DCMs 56 that represent the two devices and passes the moduleID value of the other device's DCM 56 as a parameter. (Block 102) The DCM 56 that was called then calls the data flow manager 64 to assist with making the connection and passes both the source and destination DCM moduleIDs as parameters. (Block 104) The data flow manager 64 then analyzes the source and destination IDs to determine that they are in different nodes. (Block 106) The data flow manager 64 next obtains the topology map of the network from the bus manager 7 of the source node. (Block 108) The data flow manager 64 then analyzes the topology map to find the destination node and determine if it is on the topology map. (Block 110) If the destination node is on the topology map, then the data flow manager 64 jumps to the Block 118 to determine the best route for the data transfer. If the destination node is not on the topology map, then the data flow manager 64 obtains the destination DCM from the service registry 59 in order to determine the transmission protocol for that node. (Block 112) The data flow manager 64 then finds the appropriate transmission protocol service module and sets up the appropriate conversion process. (Block 114) It is then determined if multiple transports need to be bridged. (Block 116) If multiple transports do need to be bridged then the data flow manager 64 jumps back to the Block 114 and obtains another transport conversion module. Otherwise, the data flow manager 64 then analyzes the connection paths to determine the best route for the data flow. (Block 118) The data flow manager 64 then analyzes the input data formats for the source and the destination nodes in order to determine if a conversion is necessary. (Block 120) If a conversion is necessary, the data flow manager 64 obtains the appropriate format converter from the service registry 59, based on the input and output format and sets up the conversion process. (Block 122) Otherwise, the data flow route is complete and the data transfer between the two devices can begin. (Block 124)

The bus manager 70 abstracts the underlying device interconnection mechanism, providing a common set of programming interfaces to describe the capabilities of the bus architecture. In the preferred embodiment of the present invention, the devices are connected by an IEEE 1394-1995 serial bus. For the IEEE 1394-1995 serial bus network, the bus manager 70 resides on the top of the IEEE 1394-1995 HAL layer that is provided by the host operating system 58. The bus manager 70 then helps to generalize the bus management activities up to the media manager of the present invention. The bus manager 70 notifies the client applications 48 when bus reset activity occurs by sending out bus reset notifications through the event manager 62 and providing complete information about how the environment has changed. The client applications receiving this information are provided with information about the devices which may have suddenly disappeared and the devices which have suddenly become available after the bus reset.

The bus manager 70 also provides topology maps, speed maps and other environment descriptions to client applications 48. Information from the topology map is used to build a user interface that helps the user understand the connection of the devices and how certain features may be used. This information is also used to provide automatic data routing as described above in relation to the data flow manager 64. The speed map is used to analyze the current connection scheme and to give the user helpful suggestions for improving the performance of devices on the network by rearranging the way that devices are connected. The bus manager 70 also provides atomic-level data communications services for two nodes or software modules within the nodes, to send bytes to each other in a preferred format or protocol. This protocol is built on top of those atomic communications functions.

After a bus reset or change notification of the bus, the bus manager 70 assigns new ID values to all devices which have just appeared and determines what devices have disappeared. The bus manager 70 then invokes the DCM manager 54 to create new DCMs 56 for the devices which have just appeared and posts a bus change notification to the event manager 62, which will notify all registered clients about the bus reset. This notification provides enough information for the client applications 48 to determine what devices have changed on the bus.

The transport adaptation modules 78 take care of packaging message data before it is passed on to the HAL for actual transmission to the destination device. The HAL is at the lowest layer of the media manager of the present invention. This layer provides a common programming interface upward to clients such as the DCMs 56 and any other entities that need to communicate with it. The transport adaptation modules 78 use the atomic messaging functions of the bus manager 70, as described above.

As described above, the DCMs 56 provide a protocol abstraction service, by exporting a standardized interface for device control up to the multimedia application 48. The programming interface provided by these components is divided into a common audio/video control level and a device-specific control level. The common audio/video control level provides an interface for common commands, including the basic transport control functionality, such as Play, Stop, Fast-Forward and Rewind commands. The device-specific control level provides an interface for device-specific commands, including features common to a given category of devices, such as Record for devices with recording capability, and features which are specific to a certain device or group of devices. The protocol abstraction service provided by the DCMs 56 ensures that the programming interface between the modules and the application 48 is always the same, regardless of the kind of device and the controlling protocol being used. This feature allows a great degree of flexibility for the application and the user. The DCMs 56 also provide a user input event abstraction model, so that client applications can display graphical user interface elements and send standard user event messages to the

DCM **56** as the user interacts with the graphical user interface elements, as described in U.S. Provisional Patent Application Ser. No. 60/054,199, referred to above.

The media manager of the present invention provides data flow management and other services. The media manager acts as an extension of the hosting operation system **58** and provides a variety of services to the other components of the media manager platform as well as to the client application **48**. The media manager manages and organizes the DCMs **56**. The media manager discovers and initializes the DCMs **56** which are appropriate for the applications present, while disposing of the unnecessary DCMs **56**. The media manager follows a specific sequence each time the system is booted, or any time the system could possibly change, such as when the IEEE 1394–1995 bus is reset. The media manager also provides a wrapper around the particular dynamically linked library solution that is used on the host operating system **58**. This allows the best dynamically linked library to be used to implement modules on a given operating system, while still maintaining a consistent interface to outside applications.

The media manager is also responsible for managing the flow and format of data transfer operations between the devices on the network. When managing the flow of data, the media manager will allocate and manage the appropriate buffers in a fashion independent of the operating system being used.

The media manager also provides high-level protocol management of the IEEE 1394–1995 bus environment. In order to fully support dynamic device actions such as hot plugging up to the user level, the applications and devices need to be aware of changes to the IEEE 1394–1995 bus environment. The media manager through the bus manager **70** and the event manager **62** is responsible for informing the applications and devices that bus reset activity has occurred on the IEEE 1394–1995 bus, by sending out bus reset notifications and providing complete information about how the environment has changed. The media manager also provides topology maps and other environment descriptions to the applications and devices also through the bus manager **70**. The topology map illustrates the connections between devices within the IEEE 1394–1995 network. Information derived from the topology map is used to build a human interface which helps the user understand how the devices are connected and how certain features may be used.

The application service modules **60** provide a level of services between the host operating system **58** and the application **48** in order to provide basic functionality for the application **48** independent of the specific operating system being used. This functionality includes providing memory allocation and disposal routines which are more robust than the basic functions available in most operating systems and providing device configuration and control modules which are self-contained, stand-alone modules for providing all user interface and interaction management when invoked.

The transport adaptation modules **78** provide a common programming interface to the device control modules **50** and to the application **48**, taking care of bringing the protocol capabilities up through the host operating system **58**. The internal design and implementation of the system level interface block **50** takes advantage of the specific host operating system architecture being used in order to realize the IEEE 1394–1995 functions available to the application **48**.

The media manager platform of the present invention includes the DCMs **56**, the application service modules **60** and the system level interface for IEEE 1394–1995 bus protocol provided by the transport adaptation modules **78**. During normal operation, the application **48** will communicate with all of these components. When communicating with the DCMs **56**, the application **48** will use a single programming interface. When communicating with the application service modules **60**, the application will also use a single programming interface.

A client application **48**, as described above and illustrated in FIGS. **3** and **4**, is an entity which resides above all the other components in terms of the architecture of the media manager platform of the present invention. For the completion of a majority of its required tasks, the application **48** will communicate with the DCMs **56** and the application service modules **60** which are present via the local messenger. For the times when it is necessary, the application **48** has access to the lower levels of the architecture through the host operating system **58**.

Upon startup of the client application **48**, the client application **48** must initialize and register with the media manager. The client application **48** initializes the media manager in order to make sure that the media manager is up and running and ready to serve the application **48**. The client application **48** registers with the media manager in order to give the media manager all of the necessary information for interaction with the application **48** and to register itself with the messaging system. When starting up, generally an application **48** must make sure that the host operating system has been initialized, that a minimum level of services are available and that it has the necessary amount of memory available to run. These steps are performed for the application by the media manager after the application **48** initializes the media manager.

When starting up, a client application **48** follows the steps illustrated in the flow chart of FIG. **6**. The application **48** starts up at the step **140**. After starting up, the application **48** initializes the media manager. In the preferred embodiment of the present invention the application initializes the media manager by making the following call:

```
err=SMM_Initialize
```

When initialized, the media manager will allocate the necessary memory and system services to support the application **48**.

After initialization of the media manager is complete, the application **48** then registers with the media manager at the step **144**. This step of registering allows the application **48** to provide the media manager with specific information that the media manager will need in order to properly support the application **48**. For example, the application **48** must provide the address of a callback routine for notification of significant events related to the environment, including IEEE 1394–1995 bus resets, asynchronous transaction completion and triggers when memory buffers have been filled with a specified amount of isochronous data. The step of registering is completed by the following instruction:

```
SonyErrorResultType SMM_RegisterClient
    (SMMClientIdentifierType* theClientID, SMM-
    BusEventNotificationUPP
    clientBusEventNotificationCallback, void*
    clientBusEventCallbackData);
```

The parameter theClientID is a unique identifier created by the media manager for the application. In future communication with the media manager, the application **48** will be required to pass this identifier back in, such as when it is closing down and unregistering with the media manager. The parameter clientBusEventNotificationCallback is an

appropriately formatted reference to the callback function that the application **48** will implement. The application **48** is not required to implement such a callback function if the application **48** does not need to know about dynamic changes which may occur to the network environment. If the application **48** does not implement this callback function, then the application will pass a NIL value for this parameter.

The parameter clientBusEventCallbackData can be any value that the application **48** will require access to in the callback routine. Normally, this value will be a pointer to a block of memory such that when the media manager invokes the callback function, it will pass this value back to the client application **48**, allowing the application **48** to then access global storage or other appropriate data.

To complete the step of registering with the media manager, the application **48** must also implement the notification callback function using the following interface:

pascal void (*SMMBusEventNotificationProcPtr)(void *clientData, SMMBusEventType busEventIndicator, SMMBusEventRecPtr busEventInfo);

The parameter clientData is the clientBusEventCallbackData parameter that was passed in to the registration function. The parameter busEventIndicator is an enumerated data type which indicates what kind of event the application is being notified of. The specified events include a bus reset, when a device is plugged into or unplugged from the network, the completion of an asynchronous transaction and when a specified buffer is full during an isochronous transfer of data. The parameter busEventInfo provides a data structure that contains relevant information for the particular event.

After completing the step of registering with the media manager, the application **48** then will obtain the available DCMs **56** at the step **146**. By obtaining the available DCMs **56**, the application **48** will know the other types of devices which are coupled within the network. This step is composed of a series of sub-steps. An iterative callback model is used as the method of data transfer for transferring the data to the application **48**. The client application **48** first gives the media manager an address of a callback function. The application **48** then enters a loop and repeatedly requests information about the next module from the media manager until there are no remaining DCMs. The media manager prepares a data structure with the necessary information and transfers it to the application **48** via the callback function. Once the information about each specific DCM **56** is received, the application **48** then copies the information which it requires. This process is repeated until all of the available DCMs **56** have been received by the application **48**. Within an alternate embodiment of the present invention, the client application queries the system registry, requesting a handle to each of the available DCMs **56**.

The preferred callback function which the application **48** must implement to obtain the available DCMs 56 is defined as follows:

void DeviceInfoCallbackRoutine(void *userData, SMMDeviceIndexType deviceIndex, SonyAvDeviceRecPtr deviceInfo)

The parameter userData of the callback function is the means of transferring data between the media manager and the application **48**. The application **48** will define its own data structure, allocate the memory for one of these structures and pass the address of that structure to the media manager. That address is then passed back in this callback function allowing the application **48** to access that data structure for the purpose of copying information into it.

The parameter deviceIndex of the callback function is the index value of the loop which the application **48** enters to obtain information about the available DCMs **56**. The loop is bounded by the number of available DCMs **56**. This parameter is passed back to the application **48** in the callback function so that the application **48** can save this parameter along with the other information passed into the callback function. This index value is useful in other calls to the media manager by the application **48**, when inquiring about a specific DCM **56**. In addition, this index value will be used when notifying the application **48** that a device has disappeared after it was unplugged or disconnected from the network. The application **48** will store this index value for each DCM **56** within a dedicated field in its private data structure.

The parameter deviceInfo of the callback function is a pointer to a data structure labeled SonyAVDeviceRec, in which the media manager stores the DCMs **56** for retrieval by the application **48**. The format of this data structure is known to both the application **48** and the media manager. Once a DCM **56** is stored within this data structure, the application **48** will then copy the appropriate information from the data structure to its own private data structure. The data structure SonyAvDeviceRec is defined in Table I below:

TABLE I

```
typedef struct SonyAVDeviceRec
    unsigned long               deviceID;                   //SMMDeviceIDType?
    unsigned long               busGeneration;
    SONY_DeviceModuleRefType    controlModuleReference;
    unsigned long               reserved1;
    unsigned long               reserved2;
} SonyAVDeviceRec,*SonyAVDeviceRecPtr,**SonyAVDeviceRecHdl;
```

The parameter deviceID is the identifier of a DCM **56** and correspondingly of a device. This identifier is used by the application **48** whenever it wants to communicate with a DCM **56** or when the application **48** requests services from the media manager regarding a specific device.

The parameter busGeneration is a value which changes after each bus reset action. After each bus reset, when devices are added or removed, certain information about the bus and the connected devices will change. Each time that the IEEE 1394–1995 bus is reset, the value of the parameter busGeneration is updated.

The parameter controlModuleReference is a reference to the DCM **56** that is associated with the specified device. This reference is used when the application **48** requires the media manager to act on its behalf in transactions with the module.

The application **48** will next request that the media manager generate a list of available DCMs **56** and the number of modules within that list using the following function call:

SonyErrorResultType SMM_FindDeviceControlModules (SMMDeviceListRefType* theDeviceList, unsigned long deviceAttributes, short* numAVDevices)

The parameter theDeviceList includes the address where the list of available DCMs 56 is stored and is generated and returned by the media manager. The application will declare a local variable of this type and pass the address of that variable to this function.

The parameter deviceAttributes includes a set of bitwise flags which the application 48 uses to specify the types of DCMs 56 which should be returned. For example, the application 48 may only wish to know about the active devices connected to the network. When certain flag values are specified the media manager will filter the list for only the devices meeting the criteria, before the list is returned to the application 48. The application 48 can specify that the list include all identifiable devices, only devices that are up and running, only devices that are plugged in but have their power switch turned off or only snoozing devices.

The parameter numAVDevices includes the number of DCMs 56 in the list that is returned to the application 48. The application 48 uses this number as the upper boundary of the iteration loop to obtain the DCMs 56.

The application 48 prepares the callback function address and then enters a loop to repeatedly call the media manager until the information on all of the DCMs 56 within the list is obtained. On each pass through the loop, the application 48 makes one call to the following function:

pascal SonyErrorResultType SMM_GetDeviceControlModuleInfo (SMMDeviceListRefType theDeviceList, SMMDeviceIndexType whichDevice, unsigned long reserved, SMMDeviceControlModuleIteratorUPP theDeviceListCallbackFunction, void *userData)

The parameter theDeviceList is the list reference that was returned from the function call FindAllDeviceControlModules( ). The parameter whichDevice specifies which of the DCMs 56 that the application 48 is requesting information about. The parameter theDeviceListCallbackFunction includes the prepared callback function address. The parameter userData is a reference to an application-defined data structure. This reference is passed back to the application 48 in the callback routine and the application 48 will then transfer any needed information from the media manager to this data structure.

The entire preferred sequence of the step to obtain the available DCMs 56 is listed in Table II below:

TABLE II

```
SMMDeviceListRefType     theDeviceList = NULL;
if (nil !=theDeviceList)
    err = SMM_FindAllDeviceControlModules(&theDeviceList,kActiveDevices + kInactiveDevices,
                                          &gNumAVDevices);
if (noErr == err)
{
    gAVDeviceList = NewHandle(0);
        //Prepare the callback function for the media manager:
    theDeviceInfoCallback = NewSMMDeviceControlModuleIteratorProc(DeviceInfoCallbackRoutine);
    if ((nil != theDeviceInfoCallback) && (nil != gAVDeviceList) )
    {
        for(loop = 0; loop < gNumAVDevices; loop++)
        {
            err = SMM_GetDeviceControlModuleInfo (theDeviceList, loop, 0,
                                          theDeviceInfoCallback, gAVDeviceList);
        }
        DisposeRoutineDescriptor(theDeviceInfoCallback);
    }
    else
            err = -1;
}
void DeviceInfoCallbackRoutine (void *userData, SMMDeviceIndexType deviceIndex, SonyAVDeviceRECPtr
deviceInfo)
{
        //Copy any information that I care about from the deviceInfo data structure
        //over to my private data referenced by userData:
    (myPrivateRecordPtr) userData->deviceID = deviceInfo->deviceID;
}
```

After the available DCMs 56 are obtained by the application 48, the application 48 will then obtain device specific information at the step 68. The DCM information returned by the media manager is system level information which includes the unique identifier for each device and protocol-specific information such as the bus generation for the IEEE 1394–1995 devices. In order to obtain the device specific information such as status, descriptive name string and an image of the device, the application 48 must communicate with the device through the appropriate DCM 56. By completing the steps illustrated in FIG. 6 and described above, the application 48 will have completed its startup routine and is now ready for operation.

While the application 48 is operating it will be handling user and system level events and messages including receiving control inputs, as well as messages from other processes, the host operating system and the media manager.

The present invention has been described in terms of specific embodiments incorporating details to facilitate the understanding of principles of construction and operation of the invention. Such reference herein to specific embodiments and details thereof is not intended to limit the scope of the claims appended hereto. It will be apparent to those skilled in the art that modifications may be made in the embodiment chosen for illustration without departing from the spirit and scope of the invention. Specifically, it will be apparent to those skilled in the art that while the preferred embodiment of the present invention is used to manage devices coupled together within an IEEE 1394–1995 serial bus structure, the present invention can also be implemented to manage devices within other bus structures.

We claim:

1. A method of managing operation of and communication between a network of devices comprising:

a. maintaining a control module for each device in the network, wherein the control module includes the capabilities of the device and any subdevices within the device and further wherein the control module is responsible for control of the device;

b. providing an interface to a user through which a task to be completed is requested by the user;

c. determining appropriate devices and subdevices required for completion of the task by searching the control modules; and

d. completing the task by instructing appropriate control modules to provide instructions to the appropriate devices and subdevices.

2. The method as claimed in claim 1 further comprising controlling data flow between the appropriate devices and subdevices.

3. The method as claimed in claim 2 further comprising:

a. obtaining a topology map of the devices within the network; and

b. determining a best route for the data flow by analyzing the topology map.

4. The method as claimed in claim 3 further comprising:

a. determining if conversion of data flowing between the appropriate devices and subdevices is necessary; and

b. converting the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

5. The method as claimed in claim 4 wherein the network is an IEEE 1394 serial bus network.

6. The method as claim din claim 1 wherein the control module is embedded in a corresponding device in the network.

7. The method as claimed in claim 1 further comprising downloading the control module for a corresponding device in the network.

8. The method as claimed in claim 1 wherein the control module is stored and executed remotely from a corresponding device in the network.

9. The method as claimed in claim 1 wherein the control module is in a platform neutral format.

10. The method as claimed 9 wherein the platform neutral format is Java.

11. The method as claimed in claim 1 wherein the control module provides automatic control of a corresponding device in the network.

12. An apparatus for controlling operation of and communication between a network of devices comprising:

a. a plurality of control modules, each representing a device in the network, wherein each control module includes capabilities of a corresponding device and any subdevices within the corresponding device and further wherein the control module is responsible for control of the device;

b. an interface configured to communicate with a user, wherein a task to be completed is requested by the user through the interface; and

c. a control circuit, coupled to the plurality of control modules, to the network and to the interface, configured to determine appropriate devices and subdevice required for completion of the task by searching the control modules and to complete the task by instructing appropriate control modules to provide instructions to the appropriate devices and subdevices.

13. The apparatus as claimed in claim 12 wherein the control circuit is further configured to control data flow between the devices within the network.

14. The apparatus as claimed in claim 13 further comprising a bus manager circuit, coupled to the control circuit, configured to obtain a topology map of the devices within the network and to determine a best route for the data flow by analyzing the topology map.

15. The apparatus as claimed in claim 14 wherein the control circuit is also configured to convert the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

16. The apparatus as claimed in claim 15 wherein the control circuit is also configured to implements pre-defined actions allowing users to access basic functionality of the devices in the network.

17. The apparatus as claimed in claim 16 wherein the control circuit is also configured to monitor and record user activity and to create custom, user-defined actions.

18. The apparatus as claimed in claim 15 wherein the network is an IEEE 1394 serial bus network.

19. The apparatus as claimed in claim 12 wherein one or more of the control modules are embedded within the corresponding devices.

20. The apparatus as claimed in claim 12 wherein one or more or the control modules are downloaded for each corresponding device.

21. The apparatus as claimed in claim 12 wherein one or more of the control modules are stored and executed remotely from the corresponding devices.

22. The apparatus as claimed in claim 12 wherein one or more of the control modules are in a platform neutral format.

23. The apparatus as claimed in claim 22 wherein the platform neutral format is Java.

24. The apparatus as claimed in claim 12 wherein one or more of the control modules provide automatic control of the corresponding devices.

25. An apparatus for controlling operation of and communication between a network of devices comprising:

a. means for representing each device in the network including the capabilities of the device and any subdevices within the device and further wherein the means for representing is responsible for control of the device;

b. means for interfacing for communicating with a user, wherein a task to be completed is requesting by the user through the interface; and

c. means for controlling coupled to the means for representing, to the network and to the means for interfacing for determining appropriate devices and subdevices required for completion of a task by searching the means for representing to provide instructions to the appropriate devices and subdevices.

26. The apparatus as claimed in claim 25 wherein the means for controlling further controls data flow between the devices within the network.

27. The apparatus as claimed in claim 25 further comprising a means for managing the network coupled to the means for controlling for obtaining a topology map of the devices within the network and determining a best route for data flow between the appropriate devices and subdevices by analyzing the topology map.

28. The apparatus as claimed in claim 25 wherein the means for controlling also converts data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

29. The apparatus as claimed in claim 25 wherein the means for controlling implements pre-defined actions allowing users to access basic functionality of the devices in the network.

30. The apparatus as claimed in claim 25 wherein the means for controlling also monitors and records user activity and creates custom, user-defined actions.

31. The apparatus as claimed in claim 25 wherein the network is an IEEE 1394 serial bus network.

32. The apparatus as claimed in claim 25 wherein the means for representing each device is embedded in a corresponding device.

33. The apparatus as claimed in claim 25 wherein the means for representing each device is downloaded for each corresponding device.

34. The apparatus as claimed in claim 25 wherein the means for representing each device is stored and executed remotely from a corresponding device.

35. The apparatus as claimed in claim 25 wherein the means for representing each device is in a platform neutral format.

36. The apparatus as claimed in claim 35 wherein the platform neutral format is Java.

37. The apparatus as claimed in claim 25 wherein the means for representing each device provides automatic control of a corresponding device.

38. A method of managing operation of and communication between a network of devices comprising:

a. maintaining a control module for each device in the network, wherein the control module includes the capabilities of the device and any subdevices within the device and further wherein the control module is responsible for control of the device;

b. providing an interface to a user through which a task to be completed is requested by the user; and

c. determining appropriate devices and subdevices required for completion of the task by searching the control modules.

39. The method as claimed in claim 38 further comprising completing the task by instructing appropriate control modules to provide instructions to the appropriate devices and subdevices.

40. The method as claimed in claim 38 further comprising controlling data flow between the appropriate devices and subdevices.

41. The method as claimed in claim 40 further comprising:

a. obtaining a topology map of the devices within the network; and

b. determining a best route for the data flow by analyzing the topology map.

42. The method as claimed in claim 38 further comprising:

a. determining if conversion of data flowing between the appropriate devices and subdevices is necessary; and

b. converting the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

43. The method as claimed in claim 38 wherein the network is an IEEE 1394 serial bus network.

* * * * *

US006233611B1

(12) **United States Patent**
Ludtke et al.

(10) **Patent No.:** **US 6,233,611 B1**
(45) **Date of Patent:** ***May 15, 2001**

(54) **MEDIA MANAGER FOR CONTROLLING AUTONOMOUS MEDIA DEVICES WITHIN A NETWORK ENVIRONMENT AND MANAGING THE FLOW AND FORMAT OF DATA BETWEEN THE DEVICES**

(75) Inventors: **Harold Aaron Ludtke**, San Jose; **Bruce Fairman**, Woodside; **Scott Smyers**, San Jose, all of CA (US)

(73) Assignees: **Sony Corporation**, Tokyo (JP); **Sony Electronics, Inc.**, Park Ridge, NJ (US)

(*) Notice: This patent issued on a continued prosecution application filed under 37 CFR 1.53(d), and is subject to the twenty year patent term provisions of 35 U.S.C. 154(a)(2).

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/075,047**

(22) Filed: **May 8, 1998**

(51) **Int. Cl.**[7] **G06F 15/177**

(52) **U.S. Cl.** **709/223**; 709/201; 709/208; 709/224; 340/824.25; 700/9; 710/8

(58) **Field of Search** 709/223, 224, 709/208, 201; 340/824.25; 700/8, 9; 710/8, 9, 10, 11, 12, 13

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,562,535	12/1985	Vincent et al.	364/200
4,633,392	12/1986	Vincent et al.	364/200

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

3812607 A1	10/1988	(DE)	G06F/1/00
0 499 394 A1	8/1992	(EP)	G06F/13/38
0 588 046 A1	3/1994	(EP)	G06F/13/38
0 745 929 A1	12/1996	(EP)	G06F/3/12
2203869	10/1988	(GB)	G06F/11/30

OTHER PUBLICATIONS

Chen, W. Y., "Emerging Home Digital Networking Needs," 1997 Fourth International Workshop on Community Networking Proceedings, Sep. 11–12, pp. 7–12, Doc. No. XP002126574.

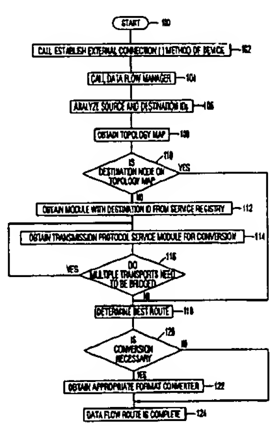
(List continued on next page.)

Primary Examiner—Dung C. Dinh
Assistant Examiner—Abdullahi E. Salad
(74) *Attorney, Agent, or Firm*—Haverstock & Owens LLP

(57) **ABSTRACT**

A media manager provides data flow management and other services for client applications on devices coupled together within a network. Preferably, these devices are coupled together within an IEEE 1394-1995 serial bus network. A device control module is generated for each available device for providing an abstraction for all of the capabilities and requirements of the device including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfers between the devices on the network. Through an interface, a user accesses the media manager and enters functions which are to be completed using the devices coupled together on the network. If the appropriate devices are available, the media manager controls and manages the completion of the requested task. If the appropriate devices are not available, but the required subdevices are available in multiple devices, the media manager forms a virtual device from subdevices in multiple devices in order to complete the requested task. Once the appropriate devices and subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from one format into another format. If necessary, the media manager will also control the format conversion during the data transfer operation. The media manager also provides network enumeration and registry searching capabilities for client applications to find available services, physical devices and virtual devices.

40 Claims, 6 Drawing Sheets



U.S. PATENT DOCUMENTS

5,289,460		2/1994	Drake, Jr. et al.	370/17
5,394,556		2/1995	Oprescu	395/800
5,457,446		10/1995	Yamamoto	340/825.24
5,574,965		11/1996	Welmer	455/3.2
5,606,664		2/1997	Brown et al.	395/200.1
5,621,662	*	4/1997	Humpheries et al.	364/550
5,621,901		4/1997	Morriss et al.	395/306
5,715,475		2/1998	Munson et al.	395/830
5,724,272	*	3/1998	Mitchell et al.	364/579
5,724,517		3/1998	Cook et al.	395/200.57
5,793,366	*	8/1998	Mano et al.	345/329
5,809,249		9/1998	Julyan	395/200.53
5,815,082		9/1998	Welmer	340/825.07
5,815,678		9/1998	Hoffman et al.	395/309
5,920,479	*	7/1999	Sojoodi et al.	364/191
5,940,387	*	8/1999	Humpleman	370/352
5,963,726	*	10/1999	Rust et al.	395/500

OTHER PUBLICATIONS

Hoffman, G. et al., "IEEE 1394: A Ubiquitous Bus," Digest of Papers of the Computer Society Computer Conference (Spring) Compcon, U.S., Los Alamitos, IEEE Comp. Soc. Press, vol. CONF. 40, Mar. 5, 1995, pp.334–338, Doc. No. XP000545446, ISBN: 0–7803–2657–1.

Miners, R. F. et al., "Dave: A Plug–and–Play Model for Distributed Multimedia Application Development," Proceedings ACM Multimedia 94, pp. 22–28, Doc. No. XP002125719.

Wright, M., "Pretenders, Contenders, or Locks for Ubiquitous Desktop Deployment?" EDN Electrical Design News, U.S., Cahners Publishing Co., Newton, Massachusetts, vol. 41, No. 9, Apr. 25, 1996, pp. 79–80, 82, 84, 86, Doc. No. XP000592137, ISSN: 0012–7515.

IEEE, "1394–1995 Standard for a High Performance Serial Bus," 1995, USA.

Wickelgren, Ingrid J., "The Facts About FireWire," IEEE Spectrum, vol. 34, No. 4, Apr. 1997, pp. 19–25, USA.

Michael Teener et al., "A Bus on Diet–The Serial Bus Alternative An Introduction to the P1394 High Performance Serial Bus" pp. 316–321.

Julia L. Heeter, "Asynchronous Transfer Mode, " Dec. 12, 1995.

R. H. J. Bloks, "The IEEE–1394 High Speed Serial Bus" Philips J. Res. 50, pp. 209–216, 1996.

* cited by examiner

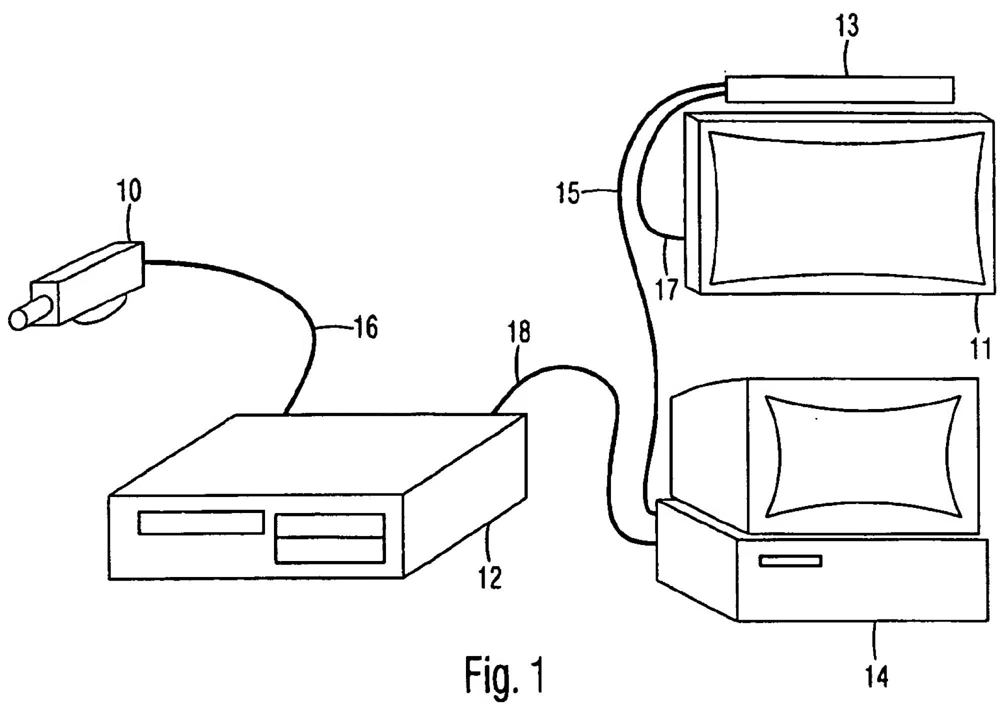


Fig. 1

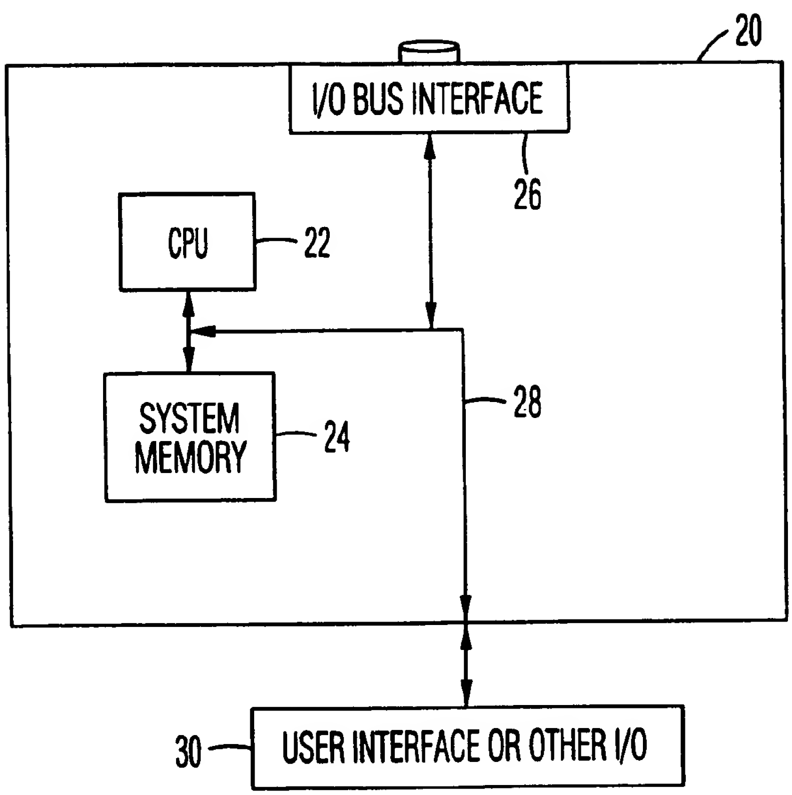

20
I/O BUS INTERFACE
26
CPU
22
28
SYSTEM MEMORY
24
30
USER INTERFACE OR OTHER I/O

Fig. 2

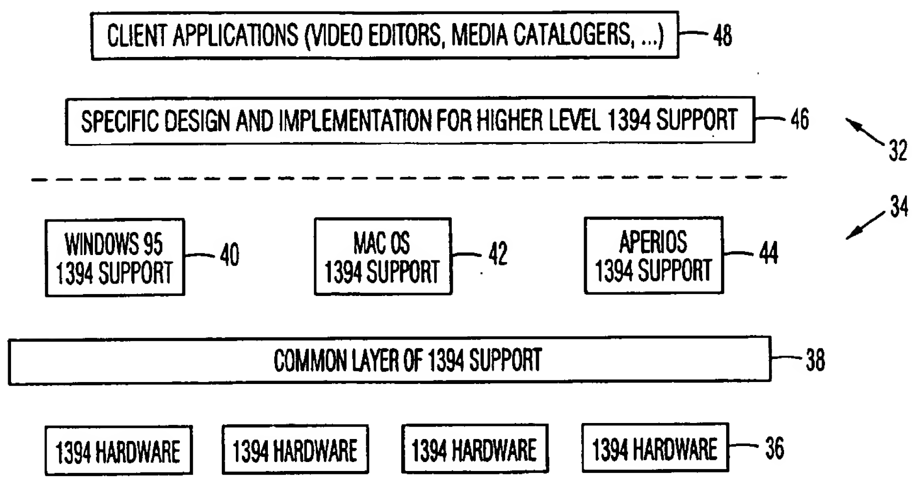

CLIENT APPLICATIONS (VIDEO EDITORS, MEDIA CATALOGERS, ...)
48
SPECIFIC DESIGN AND IMPLEMENTATION FOR HIGHER LEVEL 1394 SUPPORT
46
32
34
WINDOWS 95 1394 SUPPORT
40
MAC OS 1394 SUPPORT
42
APERIOS 1394 SUPPORT
44
COMMON LAYER OF 1394 SUPPORT
38
1394 HARDWARE
1394 HARDWARE
1394 HARDWARE
1394 HARDWARE
36

Fig. 3

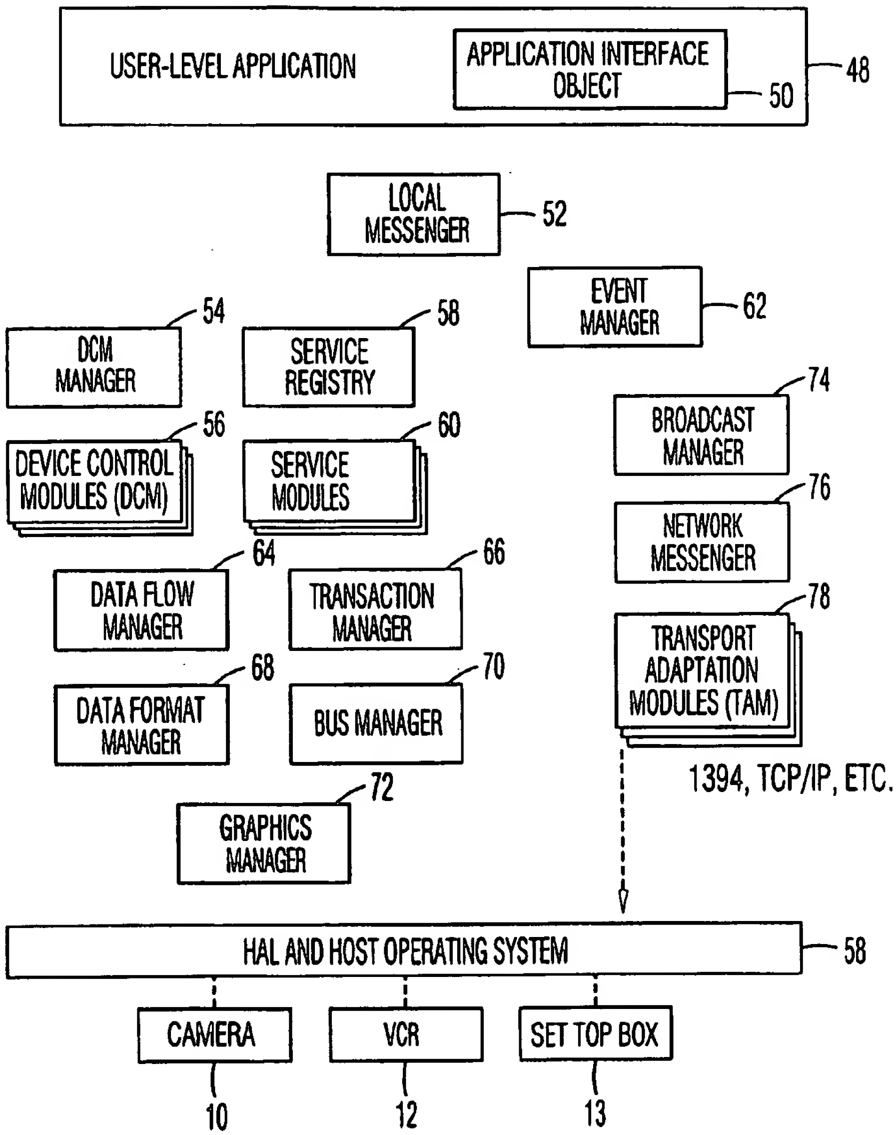

USER-LEVEL APPLICATION
APPLICATION INTERFACE OBJECT
48
50
LOCAL MESSENGER
52
EVENT MANAGER
62
54
DCM MANAGER
58
SERVICE REGISTRY
74
BROADCAST MANAGER
56
DEVICE CONTROL MODULES (DCM)
60
SERVICE MODULES
76
NETWORK MESSENGER
64
DATA FLOW MANAGER
66
TRANSACTION MANAGER
78
TRANSPORT ADAPTATION MODULES (TAM)
68
DATA FORMAT MANAGER
70
BUS MANAGER
1394, TCP/IP, ETC.
72
GRAPHICS MANAGER
HAL AND HOST OPERATING SYSTEM
58
CAMERA
VCR
SET TOP BOX
10
12
13

Fig. 4

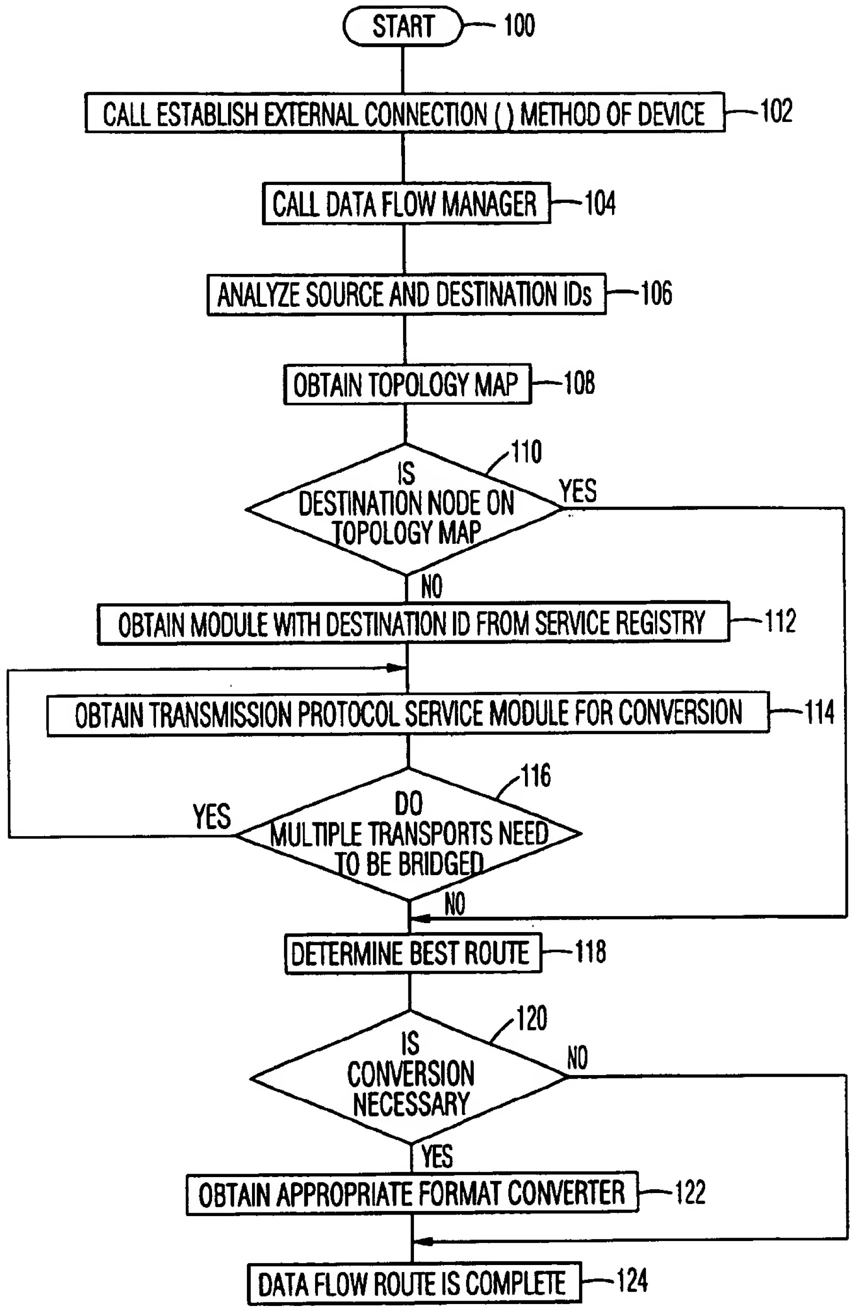

START
100
CALL ESTABLISH EXTERNAL CONNECTION ( ) METHOD OF DEVICE
102
CALL DATA FLOW MANAGER
104
ANALYZE SOURCE AND DESTINATION IDs
106
OBTAIN TOPOLOGY MAP
108
IS DESTINATION NODE ON TOPOLOGY MAP
110
YES
NO
OBTAIN MODULE WITH DESTINATION ID FROM SERVICE REGISTRY
112
OBTAIN TRANSMISSION PROTOCOL SERVICE MODULE FOR CONVERSION
114
DO MULTIPLE TRANSPORTS NEED TO BE BRIDGED
116
YES
NO
DETERMINE BEST ROUTE
118
IS CONVERSION NECESSARY
120
NO
YES
OBTAIN APPROPRIATE FORMAT CONVERTER
122
DATA FLOW ROUTE IS COMPLETE
124

Fig. 5

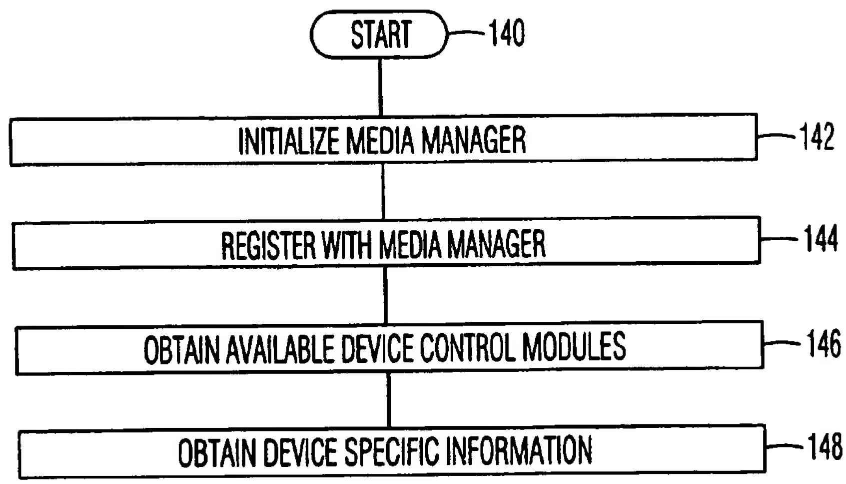

START
140
INITIALIZE MEDIA MANAGER
142
REGISTER WITH MEDIA MANAGER
144
OBTAIN AVAILABLE DEVICE CONTROL MODULES
146
OBTAIN DEVICE SPECIFIC INFORMATION
148

Fig. 6

MEDIA MANAGER FOR CONTROLLING AUTONOMOUS MEDIA DEVICES WITHIN A NETWORK ENVIRONMENT AND MANAGING THE FLOW AND FORMAT OF DATA BETWEEN THE DEVICES

FIELD OF THE INVENTION

The present invention relates to the field of managing applications and devices within a network environment. More particularly, the present invention relates to the field of managing the operation of and the communication between devices within a network environment.

BACKGROUND OF THE INVENTION

The IEEE 1394-1995 standard, "1394-1995 Standard For A High Performance Serial Bus," is an international standard for implementing an inexpensive high-speed serial bus architecture which supports both asynchronous and isochronous format data transfers. Isochronous data transfers are real-time transfers which take place such that the time intervals between significant instances have the same duration at both the transmitting and receiving applications. Each packet of data transferred isochronously is transferred in its own time period. An example of an ideal application for the transfer of data isochronously would be from a video recorder to a television set. The video recorder records images and sounds and saves the data in discrete chunks or packets. The video recorder then transfers each packet, representing the image and sound recorded over a limited time period, during that time period, for display by the television set. The IEEE 1394-1995 standard bus architecture provides multiple channels for isochronous data transfer between applications. A six bit channel number is broadcast with the data to ensure reception by the appropriate application. This allows multiple applications to simultaneously transmit isochronous data across the bus structure. Asynchronous transfers are traditional data transfer operations which take place as soon as possible and transfer an amount of data from a source to a destination.

The IEEE 1394-1995 standard provides a high-speed serial bus for interconnecting digital devices thereby providing a universal I/O connection. The IEEE 1394-1995 standard defines a digital interface for the applications thereby eliminating the need for an application to convert digital data to analog data before it is transmitted across the bus. Correspondingly, a receiving application will receive digital data from the bus, not analog data, and will therefore not be required to convert analog data to digital data. The cable required by the IEEE 1394-1995 standard is very thin in size compared to other bulkier cables used to connect such devices. Devices can be added and removed from an IEEE 1394-1995 bus while the bus is active. If a device is so added or removed the bus will then automatically reconfigure itself for transmitting data between the then existing nodes. A node is considered a logical entity with a unique address on the bus structure. Each node provides an identification ROM, a standardized set of control registers and its own address space.

Media devices are being equipped with network interfaces allowing them to become part of a network such as the IEEE 1394-1995 serial bus network. In a home audio/video network incorporating such autonomous media devices it is possible that one or more such devices will be coupled together in a network with a personal computer, settop box or other device including a microprocessor. Currently, there is a lack of available interfaces and control applications

which will efficiently manage the interaction and operation of the autonomous devices within such a network configuration. What is needed is an interface which allows a controlling device within a network configuration to efficiently control communications between the devices and the operation of the devices within the network. What is further needed is an interface which allows a controlling device within a network configuration to maximize the availability of devices within a network for completion of tasks and operations.

SUMMARY OF THE INVENTION

A media manager provides data flow management and other services for client applications on devices coupled together within a network. Preferably, these devices are coupled together within an IEEE 1394-1995 serial bus network. A device control module is generated for each available device for providing an abstraction for all of the capabilities and requirements of the device including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfers between the devices on the network. Through an interface, a user accesses the media manager and enters functions which are to be completed using the devices coupled together on the network. If the appropriate devices are available, the media manager controls and manages the completion of the requested task. If the appropriate devices are not available, but the required subdevices are available in multiple devices, the media manager forms a virtual device from subdevices in multiple devices in order to complete the requested task. Once the appropriate devices and subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from one format into another format. If necessary, the media manager will also control the format conversion during the data transfer operation. The media manager also provides network enumeration and registry searching capabilities for client applications to find available services, physical devices and virtual devices.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an IEEE 1394-1995 serial bus network including a video camera, a video cassette recorder, a computer, a television and settop box.

FIG. 2 illustrates a block diagram of a hardware system resident in each device implementing the media manager of the present invention.

FIG. 3 illustrates a block diagram of the architecture of the media manager platform of the present invention.

FIG. 4 illustrates a detailed block diagram of the architecture of the media manager platform of the present invention.

FIG. 5 illustrates a flow diagram of the steps involved in setting up and data transfer between two devices using the media manager of the present invention.

FIG. 6 illustrates a flow diagram followed by a client application during startup.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The media manager of the present invention provides data flow management and other services for physical devices within a network. A physical device is a product sold as a separate component by a vendor. Examples of physical

devices include televisions, video cassette recorders, personal computers, video cameras, CD-Rom players and the like. Many other examples are also well known and commercially available. Each physical device includes a number of subdevices. For example, a typical commercially available video camera includes multiple subdevices, implementing different functionalities, such as the camera and the video player.

Preferably, these physical devices are coupled together within an IEEE 1394-1995 serial bus network. A device control module (DCM) is generated for each available device and subdevice. Each DCM provides an abstraction for all of the capabilities and requirements of each physical device, including the appropriate control protocol, physical connections and connection capabilities for the device. The media manager also manages the flow and format of data transfer operations between the physical devices on the network, including converting the data into a different format during the data transfer operation.

Through an interface, a user accesses the media manager and enters functions and tasks which are to be completed using the physical devices within the network. If the appropriate physical devices are available and are not otherwise being used, the media manager controls and manages the completion of the requested task. If the appropriate physical devices are not available, the media manager attempts to create a virtual device from available subdevices or components within devices in order to complete the requested task. Once the appropriate physical devices and/or subdevices are assigned to a task, the media manager determines if the data to be transmitted needs to be converted from the format of the source device to the format of the receiving device. If a conversion is necessary, the media manager also controls that operation while controlling the data transfer operation.

FIG. 1 illustrates an exemplary system including a video camera **10**, a video cassette recorder **12**, a computer **14**, a settop box **13** and a television **11** connected together by the IEEE 1394-1995 cables **15**, **16** and **18**. The IEEE 1394-1995 cable **16** couples the video camera **10** to the video cassette recorder **12**, allowing the video camera **10** to send data to the video cassette recorder **12** for recording. The IEEE 1394-1995 cable **18** couples the video cassette recorder **12** to the computer **14**, allowing the video cassette recorder **12** to send data to the computer **14** for display. The IEEE 1394-1995 cable **15** couples the settop box **13** to the computer **14**. The settop box **13** is also coupled to the television **11** by the cable **17**.

This configuration illustrated in FIG. **1** is exemplary only. It should be apparent that an audio/video network could include many different combinations of physical components. The physical devices within such an IEEE 1394-1995 network are autonomous devices, meaning that in an IEEE 1394-1995 network, as the one illustrated in FIG. **1**, in which a computer is one of the devices, there is not a true "master-slave" relationship between the computer and the other devices. In many IEEE 1394-1995 network configurations, a computer may not even be present. Even in such configurations, the devices within the network are fully capable of interacting with each other on a peer basis.

A block diagram of a hardware system resident in the managing device for implementing the media manager of the present invention is illustrated in FIG. 2. In the hardware system illustrated in FIG. **2**, a printed circuit board **20** is coupled to a user interface **30**. The printed circuit board **20** includes a central processing unit (CPU) **22** coupled to system memory **24** and to an I/O bus interface **26** by the system bus **28**. The use of the term 'CPU' is not intended to imply that such a system must be a general purpose computing circuit. Rather, this circuit could be implemented with a general purpose controller or special purpose circuit. The user interface **30** is also coupled to the system bus **28**. The user interface **30** is subsystem specific, but preferably includes at least an infrared remote control device and a display. Alternatively, the user interface **30** also includes other I/O devices for communicating with a user of the subsystem.

In the preferred embodiment of the present invention, the media manager is included within a device such as a television or a computer with display, in order to facilitate smooth interaction with the user. However, it should be apparent that the media manager of the present invention can also be implemented on any other capable device which includes the components necessary for providing an interface to the user. In order to implement the media manager of the present invention, each component in which it is implemented will include a hardware system such as the system illustrated in FIG. **2**. The CPU **22** within such a device is used to execute the application program instructions. The media manager of the present invention is then used to manage communications and operations within the network. The user accesses the media manager through the interface provided at the controlling device. Through this interface the user can monitor the operation and status of the network and the devices within the network. The user can also control the devices and request tasks to be completed through this interface. An example of these tasks include playing a recorded tape on the VCR **12** and displaying the output from the VCR **12** on the television **11**. The media manager of the present invention also manages data transfer operations and tasks requested at the individual devices.

A block diagram of the architecture of the media manager platform of the present invention is illustrated in FIG. **3**. This architecture is divided into a so-called upper portion **32** and a lower portion **34**. The lower portion **34** preferably includes the IEEE 1394-1995 bus interface and functionality support across the most common commercial operating systems currently available. The upper portion **32** includes components which bring together the underlying IEEE 1394-1995 bus support and add in a number of features and enhancements which are provided to the client applications and therefore to the user. The upper portion **32** includes the block **46**, which provides specific design and implementation for higher level IEEE 1394-1995 bus support, and the block **48**, which includes and provides interfaces to the various client applications. The lower portion **34** includes the blocks **40**, **42** and **44** which provide support for the most common operating systems, including Windows 95®, Macintosh® and Aperios™ operating systems, respectively. Support for any general operating system, such as OS**9**, is also provided. The lower portion **34** also includes the block **38**, which provides the common layer of IEEE 1394-1995 support, and the blocks **36** which provide the actual physical IEEE 1394-1995 bus interfaces to other devices coupled to the controlling device.

The media manager platform provides an open and flexible architecture in order to efficiently integrate personal computers and other autonomous devices into a network configuration and effectively manage the necessary data transfer operations between those devices. The lower portion **34** of the architecture has been designed to support the underlying technology at the lowest levels, which allows the higher levels to support more general modules and functional descriptions.

A more detailed block diagram of the architecture of the media manager platform of the present invention is illustrated in FIG. 4. The multimedia or user-level application **48** sits at the top of the architecture and makes use of the services provided by the media manager. The multimedia application **48** is an application or other client of the media manager platform of the present invention. The architectural components within the media manager manage the protocol specifics and export a simpler programming interface up to the application **48**. Issues such as timing, buffer management, bus management and communication protocol are hidden behind these simple functional interfaces. The application **48** also has access to the lower layers of the architecture and will of course be able to communicate directly with the hardware adaptation layer (HAL) and the host operating system **58**. The host operating system is coupled to the other devices within the network, such as the camera **10**, the VCR **12** and the settop box **13**. For illustration purposes, in this configuration the media manager of the present invention is implemented on the computer system **14** of FIG. **1**.

The application interface object **50** serves as a proxy for the client application **48** within the media manager environment. An applications programming interface is provided to allow the client applications **48** to access the basic services of the media manager. Access to more detailed or specific functionality provided by certain programming interfaces is also provided through the application interface object **50** which provides the application **48** access to the local messenger **52**.

The local messenger **52** is one component of the messaging system integrated into the media manager. Preferably, this messaging system is the AV Messenger system. The local messenger **52** is the central hub of communications between all objects on a given node, when those objects exist in separate execution spaces. Essentially, the local messenger **52** is the inter-application communication model which is provided by the host operating system **58**. The local messenger **52** is the bottleneck through which all messages between software modules pass. To achieve scriptability, the local messenger **52** logs all messages as they pass through, keeping an internal database of all messages and their relevant data including address of destination, parameters, address of response and optionally, a time stamp for time-based scripting.

The service registry **59** includes a reference to all addressable entities within the media manager **71**. This registry includes a reference for each device control module (DCM) **56**, the DCM Manager **54**, the data flow manager **64**, the transaction manager **66**, the data format manager **68**, the bus manager **70** and the graphics manager **72**. The service registry **59** also contains any number of service modules **60**, as will be described below. The service registry **59** also contains a service registry database including references for all of the objects that are local to its node and at specific times references to remote objects as well. Each entry in the database refers to an addressable module and includes attached attributes, some of which are common to all entries and others which are specific to a type of module. Common attributes include such things as the module name and the local ID. Module-specific attributes will vary by the type of module. Once the entry exists in the service registry database, any number of attributes can be added to an entry. When a client application searches the database, the application specifies a set of attributes to match and the service registry **59** searches the database, finding and returning all entries which match the specified criteria. If multiple candidates are found during this search, the service registry **59** will provide a list reference to the client application **48**. The client application can then examine each of the candidate items in the list, to determine the items of interest.

A client application **48** may have multiple outstanding search lists, each representing the results of a different search criteria. When the client application **48** needs to update a search list, in response to an event such as a bus reset in which different devices may be available, the application **48** passes the list reference back to the service registry **59** when making a search call. This allows the service registry **59** to update the existing list object, rather than disposing of it and reallocating a new one.

The service modules **60** are modules which can be called to perform some set of services. The service modules **60** perform a variety of services for client applications, including such services as data format, transport and control protocol translation.

The DCM manager **54** is responsible for handling the group of DCMs **56** on its local node or for the devices within the controlling device's network. These responsibilities include the tasks of discovering, instantiating and disposing of all possible DCM candidates that are available to a given system. In addition, the DCM manager **54** communicates with other DCM managers on remote nodes, if any, to arbitrate for network-wide device and subdevice resource allocation and management.

The DCM manager **54** works with the underlying operating system services to get a raw list of available device node handles. The DCM manager **54** also provides an application programming interface for the client application **48** to discover what subdevices, represented by DCMs **56**, or other services are available within the devices on the network. A DCM **56** represents a device or subdevice available for allocation by the DCM manager **54**. A DCM **56** can represent a physical device or a virtual device formed of subparts from different physical devices. The other available services are represented by virtual DCMs **56**, which will be discussed below. The available DCMs will be dynamic, depending on the available physical devices on the IEEE 1394-1995 serial bus.

For each node, the DCM manager **54** does enough work to determine that it should create a DCM **56**. This is done for all media-related devices which will be managed under the umbrella of the media manager of the present invention. For each media-related entity, the DCM manager **54** creates a generic DCM **56**. Each DCM **56** then has the responsibility to make itself more device-specific, as will be described below.

Device-specific DCMs provided from manufacturers can also be added into the DCMs **56**. Device-specific DCMs can come from a variety of sources including an embedded read-only memory (ROM) within the device or some other storage mechanism such as the header of a disc or tape. The device-specific DCM may also be downloaded from an internet site or via a direct modem connection, if such capabilities are accessible to the media manager, or supplied from a disk or other storage medium. These alternatives are discussed in detail in the U.S. patent application Ser. No. 06/092703, filed on Jul. 4, 1998 (pending) and entitled "A METHOD AND APPARATUS FOR INCLUDING SELF-DESCRIBING INFORMATION WITHIN DEVICES," which is hereby incorporated by reference.

The DCM manager **54** is responsible for adding and disposing DCMs **56** at the appropriate times, and notifying clients that DCMs **56** have been added or removed. The

DCM manager **54** is also responsible for coordinating complex services among multiple DCMs **56**. These complex services, such as command queuing of complex operations, require the DCM manager **54** to coordinate with multiple DCMs **56** to carry out these operations.

The DCMs **56** each provide a device modeling and control protocol abstraction service by exporting a standardized interface for device control up to the client application **48**. The programming interface for device control provided by the DCMs **56** is divided into common A/V control and device-specific A/V control. The common A/V control commands are common to virtually all devices having audio-visual capabilities. The basic transport control functionality such as Play, Stop, Fast Forward and Rewind commands are included here. The device-specific A/V control commands include features common to a given category of A/V devices, such as the Record command for devices with recording capability, and features that are specific to a certain model or group of devices. The information for device-specific functionality can either be built into a device-specific DCM which is embedded in the device itself, using the self-describing data structures mentioned previously, or it can be downloaded as a software upgrade from the internet.

The media manager of the present invention employs protocol abstraction which means that the programming interface between the modules and the application is the same, regardless of the kind of device and the controlling protocol being used. Accordingly, the application will use the same source code and message to cause an IEEE 1394-1995 VCR to record as it would use to cause a Video System Control Architecture (VISCA) VCR to record. This is true for the common A/V control commands and the category-specific control commands; features that are truly specific to a particular device will have a unique programming interface.

The DCM **56** is the mechanism by which self-describing data from a device is downloaded and presented to the user. This requires the DCM **56** to analyze the self-describing information, by downloading and integrating modules and managing the presentation of this information to the user through a host application. This allows a user to configure and control both the well-known and device-specific functionality of devices on the network through the media manager interface. The preferred presentation of user interface data and access to custom functions of the device is described in the U.S. Provisional Patent Application Ser. No. 60/054,199, filed on Jul. 30, 1997 and entitled "METHOD FOR DESCRIBING THE HUMAN INTERFACE FEATURES AND FUNCTIONALITY OF AV/C-BASED DEVICES," which is hereby incorporated by reference.

Together, the DCM manager **54** and the DCMs **56** perform command queuing of AV commands to be executed, allowing the DCM **56** to deal with all device peculiarities, such as the need to perform a pre-roll to account for mechanical latency of a device. The DCM manager **54** and the DCMs **56** in coordination with other parts of the media manager also provide the ability to specify device control actions that are taken at specific times and as the result of certain conditions.

The DCMs **56** make up a large part of the overall architecture of the media manager of the present invention. The DCM **56** provides an abstraction for all of the various pieces of technology that make up an audio-visual device, such as the control protocol, physical connections and connection capabilities. A DCM **56** can also be created which does not represent a physical device, but represents a virtual device comprising a collection of functions or services that carry out a particular AV operation.

Physical devices and subdevices are individually accessible pieces of hardware. The media manager of the present invention uses accessible subdevices to support virtual devices to add enhanced capability to a network of devices. The virtual devices are logical entities which are combined from pieces of a variety of available components. Preferably, the virtual devices are created automatically when necessary to complete a requested task. Alternatively, the virtual devices are created dynamically by requesting the service from the DCM manager **54**.

An AV action is a pre-defined action or activity such as "watch television" or "record a movie," or any set of user-defined actions involving the manipulation of devices by using the DCMs **56**. The actions can be recorded for later re-use by a user. An AV action applications programming interface is a way of modeling common actions that are performed with devices in an AV network, such as viewing a recorded show, viewing a broadcast show, duplicating a tape and listening to a compact disk. For example, if a VCR is located in an upstairs bedroom within a user's house and is currently receiving a broadcast through the tuner and displaying it on a television in the bedroom, the transport mechanism within the VCR is not being used. If the user then desires to view a recorded show on the television downstairs, the media manager of the present invention will allow the user to place the video tape in the transport of the VCR in the bedroom and watch the recorded show from the tape on the downstairs television. The data representing the recorded show is transmitted from the upstairs VCR over the IEEE 1394-1995 serial bus network to the downstairs television. This data transfer operation is controlled by the media manager in the controlling device. Similar functions and virtual devices are achieved with tuners, demuxers, transports, amplifiers, processors and other components and subdevices. Accordingly, the user can maximize the functionality and capability of the devices within the network using the media manager to control their operation.

The DCM manager **54** keeps track not only of what physical devices and subdevices are being used, but also what virtual devices can be created from components and subcomponents that are currently available. The DCM manager **54** does this for all of its locally managed devices and for software services available on the host platform on which it is executing. The DCM manager **54** also provides the programming interfaces for a client application **48** to inquire about the virtual devices that can be created based on the resources available on the network, as well as what AV actions can be currently performed. The DCM manager **54** also ensures that virtual devices get added to the service registry **59** at the appropriate times.

The applications programming interface provided for the DCMs **56** is designed to allow the client applications **48** to discover what features and capabilities are available within the devices in the network and then work with those devices as necessary. This programming interface includes device control, device management, connection management and management of the self-describing device implementation. Each of the DCMs **56** have the responsibility to convert from a generic DCM to a device or protocol specific DCM by determining the type of device it is managing. This requires that the DCM examine the self-describing device data from the device, if any is present, and analyze any other information that may be available. The DCMs **56** also have the responsibility to provide access to the self-describing device

information data for the device being managed, including the device image, a product name string and functional descriptors to other devices and components. The DCM 56 is further responsible for providing a consistent interface for device control, including the complex services such as command queuing. Carrying out these commands requires coordination with the host operating system 58 for device control protocol usage, including packaging, sending, processing protocol-specific commands and responses via the protocol driver and other operating system provided support mechanisms.

Each DCM 56 also monitors the device it is controlling and provides extended notification support to the necessary components and applications. All normal events generated by the device go through the DCM 56 for the appropriate device, to the event manager 62 and to all interested client applications 48. In addition to supporting the AV/C device notification events, many situations may not be supported explicitly in either the AV/C protocol, in a given device or other control protocol. Depending on the capabilities of the device and its control and communications protocols, it is possible to provide extended support for such events which do not trigger actual event messages. The DCM 56 watches the device for this kind of activity and posts an event to the event manager 62.

Each DCM 56 is also responsible for managing themselves in terms of the outside entities which are making use of the data from the device under their control and the entities that are controlling them. This includes support for resource sharing and resource queuing. Resource queuing allows an entity to reserve a busy DCM for its use, as soon as the DCM 56 is available. As soon as the DCM 56 is available, it will then notify the entity.

The DCMs 56 also preferably maintain a presence during environmental changes allowing the DCM and clients to support both on-line and off-line states. This allows the DCMs 56 to quickly re-establish the services of a device once it comes on-line again.

Within the media manager of the present invention, the DCMs 56 are responsible for controlling the available devices and subdevices. The DCMs 56 provide access to the capabilities of the device, both general and device-specific. In an alternate embodiment of the present invention, each device, as part of the self-describing data, has an embedded DCM, ensuring that the software is always available regardless of where the device is taken. In a further alternate embodiment, the DCM for a specific device is obtained from the device manufacturer or a third party over the internet or provided on a media device, such as a floppy disk. In each of the above embodiments, the DCM 56, once downloaded can be stored in a variety of locations. Preferably, the DCM 56 is stored on the device it controls. However, the DCM 56 can also be stored in any appropriate location. In an alternate embodiment of the present invention, the DCM 56 is written in common byte or script code format, such as Java or Java Script, supported by the host platform. The DCM 56 is then uploaded to the host device and executed there.

The event manager 62 broadcasts all event notifications within the network to all interested parties. The event manager 62 acts as a central location for all modules within its node to register for notification when events occur in that node. The event manager 62 maintains an event notification list data structure which is a list of the defined event types and the destination identifiers of all devices which have registered for notification of each type of event. The devices each register with the event manager 62 for each event type that is of interest to them, providing their client identifier and a token value to be passed back to them when the event message is broadcast. An event is an actual occurrence of some action and a message from a device which is addressed to multiple destinations.

The event manager 62 does not usually generate events, but accepts and broadcasts events posted by other components with the media manager. When broadcasting events to client applications 48 in remote nodes, the event manager 62 makes use of the broadcast manager 74. The event manager 62 handles the interaction with the user and informs the appropriate devices accordingly. The event manager 62 informs the DCMs 56 of what user input is occurring through the interface at the control software level, so that the DCMs 56 can handle their physical devices appropriately. A DCM 56 controlling its device from a remote location will need to receive messages indicating what the user is doing and will need to send appropriate messages to its device. The event manager 62 supports the execution of remote DCMs 56 by means of the messaging system and well-defined event messages. The well-defined event messages include device administration, such as a new_device message generated when a device is added to the network, and user interaction. The user interaction messages support the preferred graphical user interface model as described in the U.S. Provisional Patent Application Ser. No. 60/054,199, filed on Jul. 30, 1997 and entitled "METHOD FOR DESCRIBING THE HUMAN INTERFACE FEATURES AND FUNCTIONALITY OF AV/C-BASED DEVICES," which is hereby incorporated by reference. In addition to well-defined event messages, any two DCMs or software modules can also define custom or private messages.

The graphics manager 72 manages the embedding of remote device controls into a controlling application and supports the remote presentation of self-describing data information by the DCMs 56. The graphics manager 72 provides a programming interface that allows the DCMs 56 to arbitrate for screen space and to work within a shared graphics environment. This allows the specific capabilities of a device to be presented to and accessed by the user through the interface of the controlling software.

The data format manager 68 manages the format of the data flowing between devices. This includes the ability to plug into the resident media manager for data format conversion as part of the buffer management and data format process. Most of the data format conversions are done transparently on behalf of the client application, based on knowledge of the source and destination of the data. Other data transformations require the client application 48 to set up a format conversion process. Preferably, the format conversion is performed in-line while the data is being transmitted. Alternatively, the format conversion is performed as either a pre or post processing task to transmission of the data. The data format conversion services available on a given platform are stored in the service registry 59. In addition to using the registry to find services, the data format manager 68 is responsible for instantiating the service modules and registering them with the service registry 59.

The data flow manager 64 works with the bus manager 70 to provide services to assist with routing data from source to destination, which may include many nodes in between. In the event that the source and destination device involve different data types, or are separated by a barrier, the data flow manager 64 will work with the data format manager 68 and the service registry 59 to handle automatic or requested data translation services as well. During the transfer of isochronous data, the data flow manager 64 provides buffer

allocation and management services. Buffer management includes the provision for a consistent notification mechanism to inform the client application when data is available for processing. While isochronous data is flowing into the client application **48**, various memory buffers are filled with the data. The data flow manager **64** informs the client application **48** when each buffer is filled so that it can handle the data acquisition process from the buffer. In addition, buffer management is simplified for client applications by having the appropriate service modules partition memory in a manner that is optimized for the data being captured. This includes separating the allocated memory into scan line or frame-sized segments for a stream of video data or the optimum segment sizes for raw audio and MIDI data.

A flow diagram of the steps involved in setting up a data transfer between two devices using the media manager of the present invention is illustrated in FIG. **5**. The method starts at the block **100**. When a client application **48** desires to establish a connection between two devices for a transfer of data, the application calls the EstablishExternalConnection() method of one of the two DCMs **56** that represent the two devices and passes the moduleID value of the other device's DCM **56** as a parameter. (Block **102**) The DCM **56** that was called then calls the data flow manager **64** to assist with making the connection and passes both the source and destination DCM moduleIDs as parameters. (Block **104**) The data flow manager **64** then analyzes the source and destination IDs to determine that they are in different nodes. (Block **106**) The data flow manager **64** next obtains the topology map of the network from the bus manager **7** of the source node. (Block **108**) The data flow manager **64** then analyzes the topology map to find the destination node and determine if it is on the topology map. (Block **110**) If the destination node is on the topology map, then the data flow manager **64** jumps to the Block **118** to determine the best route for the data transfer. If the destination node is not on the topology map, then the data flow manager **64** obtains the destination DCM from the service registry **59** in order to determine the transmission protocol for that node. (Block **112**) The data flow manager **64** then finds the appropriate transmission protocol service module and sets up the appropriate conversion process. (Block **114**) It is then determined if multiple transports need to be bridged. (Block **116**) If multiple transports do need to be bridged then the data flow manager **64** jumps back to the Block **114** and obtains another transport conversion module. Otherwise, the data flow manager **64** then analyzes the connection paths to determine the best route for the data flow. (Block **118**) The data flow manager **64** then analyzes the input data formats for the source and the destination nodes in order to determine if a conversion is necessary. (Block **120**) If a conversion is necessary, the data flow manager **64** obtains the appropriate format converter from the service registry **59**, based on the input and output format and sets up the conversion process. (Block **122**) Otherwise, the data flow route is complete and the data transfer between the two devices can begin. (Block **124**)

The bus manager **70** abstracts the underlying device interconnection mechanism, providing a common set of programming interfaces to describe the capabilities of the bus architecture. In the preferred embodiment of the present invention, the devices are connected by an IEEE 1394-1995 serial bus. For the IEEE 1394-1995 serial bus network, the bus manager **70** resides on the top of the IEEE 1394-1995 HAL layer that is provided by the host operating system **58**. The bus manager **70** then helps to generalize the bus management activities up to the media manager of the present invention. The bus manager **70** notifies the client applications **48** when bus reset activity occurs by sending out bus reset notifications through the event manager **62** and providing complete information about how the environment has changed. The client applications receiving this information are provided with information about the devices which may have suddenly disappeared and the devices which have suddenly become available after the bus reset.

The bus manager **70** also provides topology maps, speed maps and other environment descriptions to client applications **48**. Information from the topology map is used to build a user interface that helps the user understand the connection of the devices and how certain features may be used. This information is also used to provide automatic data routing as described above in relation to the data flow manager **64**. The speed map is used to analyze the current connection scheme and to give the user helpful suggestions for improving the performance of devices on the network by rearranging the way that devices are connected. The bus manager **70** also provides atomic-level data communications services for two nodes or software modules within the nodes, to send bytes to each other in a preferred format or protocol. This protocol is built on top of those atomic communications functions.

After a bus reset or change notification of the bus, the bus manager **70** assigns new ID values to all devices which have just appeared and determines what devices have disappeared. The bus manager **70** then invokes the DCM manager **54** to create new DCMs **56** for the devices which have just appeared and posts a bus change notification to the event manager **62**, which will notify all registered clients about the bus reset. This notification provides enough information for the client applications **48** to determine what devices have changed on the bus.

The transport adaptation modules **78** take care of packaging message data before it is passed on to the HAL for actual transmission to the destination device. The HAL is at the lowest layer of the media manager of the present invention. This layer provides a common programming interface upward to clients such as the DCMs **56** and any other entities that need to communicate with it. The transport adaptation modules **78** use the atomic messaging functions of the bus manager **70**, as described above.

As described above, the DCMs **56** provide a protocol abstraction service, by exporting a standardized interface for device control up to the multimedia application **48**. The programming interface provided by these components is divided into a common audio/video control level and a device-specific control level. The common audio/video control level provides an interface for common commands, including the basic transport control functionality, such as Play, Stop, Fast-Forward and Rewind commands. The device-specific control level provides an interface for device-specific commands, including features common to a given category of devices, such as Record for devices with recording capability, and features which are specific to a certain device or group of devices. The protocol abstraction service provided by the DCMs **56** ensures that the programming interface between the modules and the application **48** is always the same, regardless of the kind of device and the controlling protocol being used. This feature allows a great degree of flexibility for the application and the user. The DCMs **56** also provide a user input event abstraction model, so that client applications can display graphical user interface elements and send standard user event messages to the DCM **56** as the user interacts with the graphical user interface elements, as described in U.S. Provisional Patent Application Ser. No. 60/054,199, referred to above.

The media manager of the present invention provides data flow management and other services. The media manager acts as an extension of the hosting operation system **58** and provides a variety of services to the other components of the media manager platform as well as to the client application **48**. The media manager manages and organizes the DCMs **56**. The media manager discovers and initializes the DCMs **56** which are appropriate for the applications present, while disposing of the unnecessary DCMs **56**. The media manager follows a specific sequence each time the system is booted, or any time the system could possibly change, such as when the IEEE 1394-1995 bus is reset. The media manager also provides a wrapper around the particular dynamically linked library solution that is used on the host operating system **58**. This allows the best dynamically linked library to be used to implement modules on a given operating system, while still maintaining a consistent interface to outside applications.

The media manager is also responsible for managing the flow and format of data transfer operations between the devices on the network. When managing the flow of data, the media manager will allocate and manage the appropriate buffers in a fashion independent of the operating system being used.

The media manager also provides high-level protocol management of the IEEE 1394-1995 bus environment. In order to fully support dynamic device actions such as hot plugging up to the user level, the applications and devices need to be aware of changes to the IEEE 1394-1995 bus environment. The media manager through the bus manager **70** and the event manager **62** is responsible for informing the applications and devices that bus reset activity has occurred on the IEEE 1394-1995 bus, by sending out bus reset notifications and providing complete information about how the environment has changed. The media manager also provides topology maps and other environment descriptions to the applications and devices also through the bus manager **70**. The topology map illustrates the connections between devices within the IEEE 1394-1995 network. Information derived from the topology map is used to build a human interface which helps the user understand how the devices are connected and how certain features may be used.

The application service modules **60** provide a level of services between the host operating system **58** and the application **48** in order to provide basic functionality for the application **48** independent of the specific operating system being used. This functionality includes providing memory allocation and disposal routines which are more robust than the basic functions available in most operating systems and providing device configuration and control modules which are self-contained, stand-alone modules for providing all user interface and interaction management when invoked.

The transport adaptation modules **78** provide a common programming interface to the device control modules **50** and to the application **48**, taking care of bringing the protocol capabilities up through the host operating system **58**. The internal design and implementation of the system level interface block **50** takes advantage of the specific host operating system architecture being used in order to realize the IEEE 1394-1995 functions available to the application **48**.

The media manager platform of the present invention includes the DCMs **56**, the application service modules **60** and the system level interface for IEEE 1394-1995 bus protocol provided by the transport adaptation modules **78**. During normal operation, the application **48** will communicate with all of these components. When communicating with the DCMs **56**, the application **48** will use a single programming interface. When communicating with the application service modules **60**, the application will also use a single programming interface.

A client application **48**, as described above and illustrated in FIGS. **3** and **4**, is an entity which resides above all the other components in terms of the architecture of the media manager platform of the present invention. For the completion of a majority of its required tasks, the application **48** will communicate with the DCMs **56** and the application service modules **60** which are present via the local messenger. For the times when it is necessary, the application **48** has access to the lower levels of the architecture through the host operating system **58**.

Upon startup of the client application **48**, the client application **48** must initialize and register with the media manager. The client application **48** initializes the media manager in order to make sure that the media manager is up and running and ready to serve the application **48**. The client application **48** registers with the media manager in order to give the media manager all of the necessary information for interaction with the application **48** and to register itself with the messaging system. When starting up, generally an application **48** must make sure that the host operating system has been initialized, that a minimum level of services are available and that it has the necessary amount of memory available to run. These steps are performed for the application by the media manager after the application **48** initializes the media manager.

When starting up, a client application **48** follows the steps illustrated in the flow chart of FIG. **6**. The application **48** starts up at the step **140**. After starting up, the application **48** initializes the media manager. In the preferred embodiment of the present invention the application initializes the media manager by making the following call:

err=SMM_Initialize

When initialized, the media manager will allocate the necessary memory and system services to support the application **48**.

After initialization of the media manager is complete, the application **48** then registers with the media manager at the step **144**. This step of registering allows the application **48** to provide the media manager with specific information that the media manager will need in order to properly support the application **48**. For example, the application **48** must provide the address of a callback routine for notification of significant events related to the environment, including IEEE 1394-1995 bus resets, asynchronous transaction completion and triggers when memory buffers have been filled with a specified amount of isochronous data. The step of registering is completed by the following instruction:

SonyErrorResultType SMM_RegisterClient (SMMClientIdentifierType* the ClientID,
SMMBusEventNotificationUPP clientBusEventNotificationCallback, void*
clientBusEventCallbackData);

The parameter theClientID is a unique identifier created by the media manager for the application. In future communication with the media manager, the application **48** will be required to pass this identifier back in, such as when it is closing down and unregistering with the media manager. The parameter clientBusEventNotificationCallback is an appropriately formatted reference to the callback function that the application **48** will implement. The application **48** is not required to implement such a callback function if the application **48** does not need to know about dynamic changes which may occur to the network environment. If the

application **48** does not implement this callback function, then the application will pass a NIL value for this parameter.

The parameter clientBusEventCallbackData can be any value that the application **48** will require access to in the callback routine. Normally, this value will be a pointer to a block of memory such that when the media manager invokes the callback function, it will pass this value back to the client application **48**, allowing the application **48** to then access global storage or other appropriate data.

To complete the step of registering with the media manager, the application **48** must also implement the notification callback function using the following interface:

pascal void (*SMMBusEventNotificationProcPtr)(void *clientData,

SMMBusEventType busEventIndicator, SMMBusEventRecPtr busEventInfo);

The parameter clientData is the clientBusEventCallbackData parameter that was passed in to the registration function. The parameter busEventIndicator is an enumerated data type which indicates what kind of event the application is being notified of. The specified events include a bus reset, when a device is plugged into or unplugged from the network, the completion of an asynchronous transaction and when a specified buffer is full during an isochronous transfer of data. The parameter busEventInfo provides a data structure that contains relevant information for the particular event.

After completing the step of registering with the media manager, the application **48** then will obtain the available DCMs **56** at the step **146**. By obtaining the available DCMs **56**, the application **48** will know the other types of devices which are coupled within the network. This step is composed of a series of sub-steps. An iterative callback model is used as the method of data transfer for transferring the data to the application **48**. The client application **48** first gives the media manager an address of a callback function. The application **48** then enters a loop and repeatedly requests information about the next module from the media manager until there are no remaining DCMs. The media manager prepares a data structure with the necessary information and transfers it to the application **48** via the callback function. Once the information about each specific DCM **56** is received, the application **48** then copies the information which it requires. This process is repeated until all of the available DCMs **56** have been received by the application **48**. Within an alternate embodiment of the present invention, the client application queries the system registry, requesting a handle to each of the available DCMs **56**.

The preferred callback function which the application **48** must implement to obtain the available DCMs **56** is defined as follows:

void DeviceInfoCallbackRoutine(void *userData, SMMDeviceIndexType deviceIndex, SonyAvDeviceRecPtr deviceInfo)

The parameter userData of the callback function is the means of transferring data between the media manager and the application **48**. The application **48** will define its own data structure, allocate the memory for one of these structures and pass the address of that structure to the media manager. That address is then passed back in this callback function allowing the application **48** to access that data structure for the purpose of copying information into it.

The parameter deviceIndex of the callback function is the index value of the loop which the application **48** enters to obtain information about the available DCMs **56**. The loop is bounded by the number of available DCMs **56**. This parameter is passed back to the application **48** in the callback function so that the application **48** can save this parameter along with the other information passed into the callback function. This index value is useful in other calls to the media manager by the application **48**, when inquiring about a specific DCM **56**. In addition, this index value will be used when notifying the application **48** that a device has disappeared after it was unplugged or disconnected from the network. The application **48** will store this index value for each DCM **56** within a dedicated field in its private data structure.

The parameter deviceInfo of the callback function is a pointer to a data structure labeled SonyAVDeviceRec, in which the media manager stores the DCMs **56** for retrieval by the application **48**. The format of this data structure is known to both the application **48** and the media manager. Once a DCM **56** is stored within this data structure, the application **48** will then copy the appropriate information from the data structure to its own private data structure. The data structure SonyAvDeviceRec is defined in Table I below:

TABLE I

```
typedef struct SonyAVDeviceRec
unsigned long                  deviceID;   //SMMDeviceIDType?
unsigned long                  busGeneration;
SONY_DeviceModuleRefType       controlModuleReference;
unsigned long                  reserved1;
unsigned long                  reserved2;
} SonyAVDeviceRec, *SonyAVDeviceRecPtr, **SonyAvDeviceRecHdl;
```

The parameter deviceID is the identifier of a DCM **56** and correspondingly of a device. This identifier is used by the application **48** whenever it wants to communicate with a DCM **56** or when the application **48** requests services from the media manager regarding a specific device.

The parameter busGeneration is a value which changes after each bus reset action. After each bus reset, when devices are added or removed, certain information about the bus and the connected devices will change. Each time that the IEEE 1394-1995 bus is reset, the value of the parameter busGeneration is updated.

The parameter controlModuleReference is a reference to the DCM **56** that is associated with the specified device. This reference is used when the application **48** requires the media manager to act on its behalf in transactions with the module.

The application **48** will next request that the media manager generate a list of available DCMs **56** and the number of modules within that list using the following function call:

SonyErrorResultType SMM_FindDeviceControlModules (SMMDeviceListRefType* theDeviceList, unsigned long deviceAttributes, short* numAVDevices)

The parameter theDeviceList includes the address where the list of available DCMs **56** is stored and is generated and returned by the media manager. The application will declare a local variable of this type and pass the address of that variable to this function.

The parameter deviceAttributes includes a set of bitwise flags which the application **48** uses to specify the types of DCMs **56** which should be returned. For example, the application **48** may only wish to know about the active devices connected to the network. When certain flag values are specified the media manager will filter the list for only the devices meeting the criteria, before the list is returned to the application **48**. The application **48** can specify that the

list include all identifiable devices, only devices that are up and running, only devices that are plugged in but have their power switch turned off or only snoozing devices.

The parameter numAVDevices includes the number of DCMs **56** in the list that is returned to the application **48**. The application **48** uses this number as the upper boundary of the iteration loop to obtain the DCMs **56**.

The application **48** prepares the callback function address and then enters a loop to repeatedly call the media manager until the information on all of the DCMs **56** within the list is obtained. On each pass through the loop, the application **48** makes one call to the following function:

pascal SonyErrorResultType SMM_GetDeviceControlModuleInfo (SMMDeviceListRefType theDeviceList, SMMDeviceIndexType whichDevice, unsigned long reserved, SMMDeviceControlModuleIteratorUPP theDeviceListCallbackFunction, void *userData)

The parameter theDeviceList is the list reference that was returned from the function call FindAllDeviceControlModules(). The parameter whichDevice specifies which of the DCMs **56** that the application **48** is requesting information about. The parameter theDeviceListCallbackFunction includes the prepared callback function address. The parameter userData is a reference to an application-defined data structure. This reference is passed back to the application **48** in the callback routine and the application **48** will then transfer any needed information from the media manager to this data structure.

The entire preferred sequence of the step to obtain the available DCMs **56** is listed in Table II below:

TABLE II

```
SMMDeviceListRefType          theDeviceList = NULL;
if (nil != theDeviceList)
    err = SMM_FindAllDeviceControlModules(&theDeviceList, kActiveDevices + kInactiveDevices,
                                          &gNumAvDevices);
if (noErr == err)
{
    gAvDeviceList = NewHandle(0);
        //Prepare the callback function for the media manager:
    theDeviceInfoCallback = NewSMMDeviceControlModuleIteratorProc(DeviceInfoCallbackRoutine);
    if ( (nil != theDeviceInfoCallback) && (nil != gAVDeviceList) )
    {
        for(loop = 0; loop < gNumAvDevices; loop++)
        {
            err = SMM_GetDeviceControlModuleInfo (theDeviceList, loop, 0,
                                          theDeviceInfoCallback, gAVDeviceList);
        }
        DisposeRoutineDescriptor(theDeviceInfoCallback);
    }
    else
        err = -1;
}
void DeviceInfoCallbackRoutine(void *userData, SMMDeviceIndexType deviceIndex, SonyAVDeviceRECPtr
deviceInfo)
{
        //Copy any information that I care about from the deviceInfo data structure
        //over to my private data referenced by userData:
    (myPrivateRecordPtr) userData->deviceID = deviceInfo->deviceID;
}
```

After the available DCMs **56** are obtained by the application **48**, the application **48** will then obtain device specific information at the step **68**. The DCM information returned by the media manager is system level information which includes the unique identifier for each device and protocol-specific information such as the bus generation for the IEEE 1394-1995 devices. In order to obtain the device specific information such as status, descriptive name string and an image of the device, the application **48** must communicate with the device through the appropriate DCM **56**. By completing the steps illustrated in FIG. **6** and described above, the application **48** will have completed its startup routine and is now ready for operation.

While the application **48** is operating it will be handling user and system level events and messages including receiving control inputs, as well as messages from other processes, the host operating system and the media manager.

The present invention has been described in terms of specific embodiments incorporating details to facilitate the understanding of principles of construction and operation of the invention. Such reference herein to specific embodiments and details thereof is not intended to limit the scope of the claims appended hereto. It will be apparent to those skilled in the art that modifications may be made in the embodiment chosen for illustration without departing from the spirit and scope of the invention. Specifically, it will be apparent to those skilled in the art that while the preferred embodiment of the present invention is used to manage devices coupled together within an IEEE 1394-1995 serial bus structure, the present invention can also be implemented to manage devices within other bus structures.

We claim:

1. A method of managing operation of and communication between a network of devices for completing a task comprising:

a. determining appropriate devices and subdevices required for completion of the task, wherein virtual devices are formed from available subdevices if appropriate devices are not available for completion of the task; and

b. instructing the appropriate devices and subdevices to complete the task.

2. The method as claimed in claim **1** further comprising maintaining a control module for each device in the network, wherein the control module includes the capabilities of the device and any subdevices within the device and further wherein the control module is responsible for control of the device.

3. The method as claimed in claim 2 further comprising controlling data flow between the appropriate devices and subdevices.

4. The method as claimed in claim 3 further comprising:

a. obtaining a topology map of the devices within the network; and

b. determining a best route for the data flow by analyzing the topology map.

5. The method as claimed in claim 4 further comprising converting the data flowing between the appropriate devices and subdevices into a proper format, if necessary.

6. The method as claimed in claim 5 further comprising providing an interface to a user through which the task to be completed is requested.

7. The method as claimed in claim 6 wherein the control module provides user interface data to an application and responds to user events from the application.

8. The method as claimed in claim 7 wherein the network is an IEEE 1394 serial bus network.

9. The method as claimed in claim 7 wherein the control module resides in a remote device and is downloaded to a host device for execution.

10. The method as claimed in claim 7 wherein the control module resides in a local device and is executed from a native environment within the local device.

11. The method as claimed in claim 7 wherein the control module resides in a local device and is uploaded to a host device for execution.

12. An apparatus for controlling operation of and communication between a network of devices comprising:

a. an interface circuit configured to communicate with the devices within the network; and

b. a control circuit, coupled to the interface circuit, configured to determine appropriate devices and subdevices required for completion of a task and to instruct the appropriate devices and subdevices to complete the task, wherein virtual devices are formed from available subdevices if appropriate devices are not available for completion of the task.

13. The apparatus as claimed in claim 12 further comprising a plurality of control modules, each representing a device in the network, wherein each control module includes capabilities of a corresponding device and any subdevices within the corresponding device and further wherein the control module is responsible for control of the device.

14. The apparatus as claimed in claim 13 further comprising a bus manager circuit, coupled to the control circuit and to the interface circuit, configured to obtain a topology map of the devices within the network and determining a best route for data flow between the appropriate devices and subdevices by analyzing the topology map.

15. The apparatus as claimed in claim 14 wherein the control circuit is also configured to convert the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

16. The apparatus as claimed in claim 15 wherein the network is an IEEE 1394 serial bus network.

17. A method of managing operation of and communication between a network of devices comprising:

a. maintaining a control module for each device in the network, wherein the control module includes the capabilities of the device and any subdevices within the device and further wherein the control module is responsible for control of the device;

b. providing an interface to a user through which a task to be completed is requested by the user;

c. determining appropriate devices and subdevices required for completion of the task by searching the control modules, wherein virtual devices are formed from available subdevices if appropriate devices are not available for completion of the task; and

d. completing the task by instructing appropriate control modules to provide instructions to the appropriate devices and subdevices.

18. The method as claimed in claim 18 further comprising controlling data flow between the appropriate devices and subdevices.

19. The method as claimed in claim 18 further comprising:

a. obtaining a topology map of the devices within the network; and

b. determining a best route for the data flow by analyzing the topology map.

20. The method as claimed in claim 19 further comprising:

a. determining if conversion of data flowing between the appropriate devices and subdevices is necessary; and

b. converting the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

21. The method as claimed in claim 20 wherein the network is an IEEE 1394 serial bus network.

22. An apparatus for controlling operation of and communication between a network of devices comprising:

a. a plurality of control modules, each representing a device in the network wherein each control module includes capabilities of a corresponding device and any subdevices within the corresponding device and further wherein the control module is responsible for control of the device;

b. an interface configured to communicate with a user wherein a task to be completed is requested by the user through the interface; and

c. a control circuit, coupled to the plurality of control modules, to the network and to the interface, configured to determine appropriate devices and subdevices required for completion of the task by searching the control modules and to complete the task by instructing appropriate control modules to provide instructions to the appropriate devices and subdevices, wherein the control circuit also determines if the appropriate devices and subdevices are currently available for completion of the task and forms virtual devices from available subdevices to complete the task when the appropriate devices and subdevices are not currently available.

23. The apparatus as claimed in claim 22 wherein the control circuit is further configured to control data flow between the devices within the network.

24. The apparatus as claimed in claim 23 further comprising a bus manager circuit, coupled to the control circuit, configured to obtain a topology map of the devices within the network and to determine a best route for the data flow by analyzing the topology map.

25. The apparatus as claimed in claim 24 wherein the control circuit is also configured to convert the data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

26. The apparatus as claimed in claim 25 wherein the control circuit is also configured to implement pre-defined actions allowing users to access basic functionality of the devices in the network.

27. The apparatus as claimed in claim 26 wherein the control circuit is also configured to monitor and record user activity and to create custom, user-defined actions.

28. The apparatus as claimed in claim 25 wherein the network is an IEEE 1394 serial bus network.

29. An apparatus for controlling operation of and communication between a network of devices comprising:

a. means for interfacing for communicating with the devices within the network; and

b. means for controlling coupled to the means for interfacing for determining appropriate devices and subdevices required for completion of a task and instructing the appropriate devices and subdevices to complete the task, wherein virtual devices are formed from available subdevices if appropriate devices are not available for completion of the task.

30. The apparatus as claimed in claim 29 further comprising a means for representing each device in the network including the capabilities of the device and any subdevices within the device and further wherein the means for representing is responsible for control of the device.

31. The apparatus as claimed in claim 29 further comprising a means for managing the network coupled to the means for interfacing and the means for controlling for obtaining a topology map of the devices within the network and determining a best route for data flow between the appropriate devices and subdevices by analyzing the topology map.

32. The apparatus as claimed in claim 29 wherein the means for controlling also converts data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

33. The apparatus as claimed in claim 29 wherein the network is an IEEE 1394 serial bus network.

34. An apparatus for controlling operation of and communication between a network of devices comprising:

a. means for representing each device in the network including the capabilities of the device and any subdevices within the device and further wherein the means for representing is responsible for control of the device;

b. means for interfacing for communicating with a user, wherein a task to be completed is requested by the user through the interface; and

c. means for controlling coupled to the means for representing, to the network and to the means for interfacing for determining appropriate devices and subdevices required for completion of a task by searching the means for representing and completing the task by instructing the means for representing to provide instructions to the appropriate devices and subdevices, wherein the means for controlling also determines if the appropriate devices and subdevices are currently available for completion of the task and forms virtual devices from available subdevices to complete the task when the appropriate devices and subdevices are not currently available.

35. The apparatus as claimed in claim 34 wherein the means for controlling further controls data flow between the devices within the network.

36. The apparatus as claimed in claim 34 further comprising a means for managing the network coupled to the means for controlling for obtaining a topology map of the devices within the network and determining a best route for data flow between the appropriate devices and subdevices by analyzing the topology map.

37. The apparatus as claimed in claim 34 wherein the means for controlling also converts data flowing between the appropriate devices and subdevices into a proper format if data conversion is necessary.

38. The apparatus as claimed in claim 34 wherein the means for controlling implements pre-defined actions allowing users to access basic functionality of the devices in the network.

39. The apparatus as claimed in claim 34 wherein the means for controlling also monitors and records user activity and creates custom, user-defined actions.

40. The apparatus as claimed in claim 34 wherein the network is an IEEE 1394 serial bus network.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,233,611 B1
DATED : May 15, 2001
INVENTOR(S) : Harold Aaron Ludtke et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 6,
Line 61, please delete "06/092703, filed on Jul." and insert therefor -- 09/092703, filed on Jun. --.

Claim 18,
Line 1, please delete "claim 18" and insert therefor -- claim 17 --.

Signed and Sealed this

Twenty-seventh Day of November, 2001

Attest:

Nicholas P. Godici

NICHOLAS P. GODICI

Attesting Officer *Acting Director of the United States Patent and Trademark Office*

US006038625A

United States Patent [19]

Ogino et al.

[11] Patent Number: 6,038,625

[45] Date of Patent: Mar. 14, 2000

[54] METHOD AND SYSTEM FOR PROVIDING A DEVICE IDENTIFICATION MECHANISM WITHIN A CONSUMER AUDIO/VIDEO NETWORK

[75] Inventors: **Hiroshi Ogino**, Sunnyvale; **Feng (Frank) Zou**, Milpitas, both of Calif.

[73] Assignees: **Sony Corporation of Japan**, Tokyo, Japan; **Sony Electronics, Inc.**, Park Ridge, N.J.

[21] Appl. No.: **09/003,111**

[22] Filed: **Jan. 6, 1998**

[51] **Int. Cl.**[7] **G06F 13/00**

[52] **U.S. Cl.** **710/104**; 710/8; 710/9; 710/10; 709/302; 712/1; 712/208

[58] **Field of Search** 710/43, 104, 8, 710/9, 10, 2; 712/208, 1; 709/302, 300, 220, 221, 222; 348/552

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,893,199	1/1990	Okada	360/48
5,265,241	11/1993	Arnold et al.	710/15
5,278,961	1/1994	Mueller	711/206
5,420,573	5/1995	Tanaka et al.	340/925.24
5,537,605	7/1996	Teece	395/800
5,675,830	10/1997	Satula	710/9
5,784,702	7/1998	Greenstein et al.	711/173
5,815,678	9/1998	Hoffman et al.	710/129
5,848,247	12/1998	Matsui et al.	710/104

FOREIGN PATENT DOCUMENTS

0739110A2	10/1996	European Pat. Off.	H04L 12/24
2767795	3/1989	Japan	H04L 5/22
404038549A	2/1992	Japan	G06F 13/00

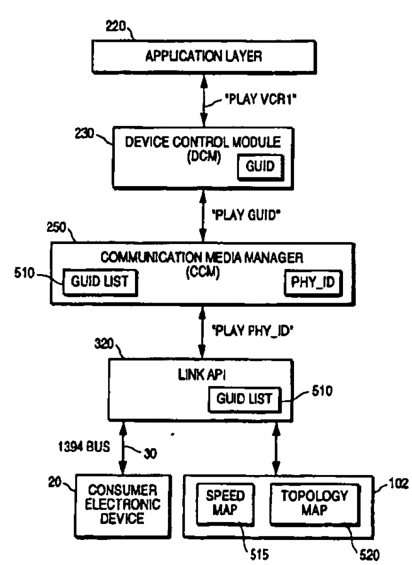
Primary Examiner—Xuan M. Thai
Attorney, Agent, or Firm—Wagner, Murabito & Hao LLP

[57] **ABSTRACT**

A method and system for providing a device identification mechanism within a consumer electronics based audio/video network. Several consumer electronics products, e.g., television, VCR, tuner, set-top box (e.g., intelligent receiver/decoder, IRD), DVTRs, PCs, DVD players (digital video disk), etc., can be coupled within the network to communicate together via a standard bus (e.g., IEEE 1394 serial communication bus). In one embodiment, the HAVI network offers unique advantages consumer electronic vendors because the architecture offers for the home network many of the advantages of existing computer system networks. Specifically, interconnected devices can share resources and provide open, well defined APIs that allow ease of development for third party developers. The present invention provides a mechanism whereby a global unique identifier (GUID) is associated with each device of the HAVI network. A low level driver constructs a GUID list of each device on the HAVI network. The order of the GUID entries in the GUID list (e.g., the index) matches the physical identifiers assigned to the devices by the 1394 serial bus. Although the physical identifiers can change on bus reset, the GUID values are constant and are used for device communication. Speed map and topology map information is maintained based on the physical identifier information and therefore translations between GUIDs and physical identifiers are efficiently performed by the present invention when referencing speed map and topology information for an application.

22 Claims, 26 Drawing Sheets

305b



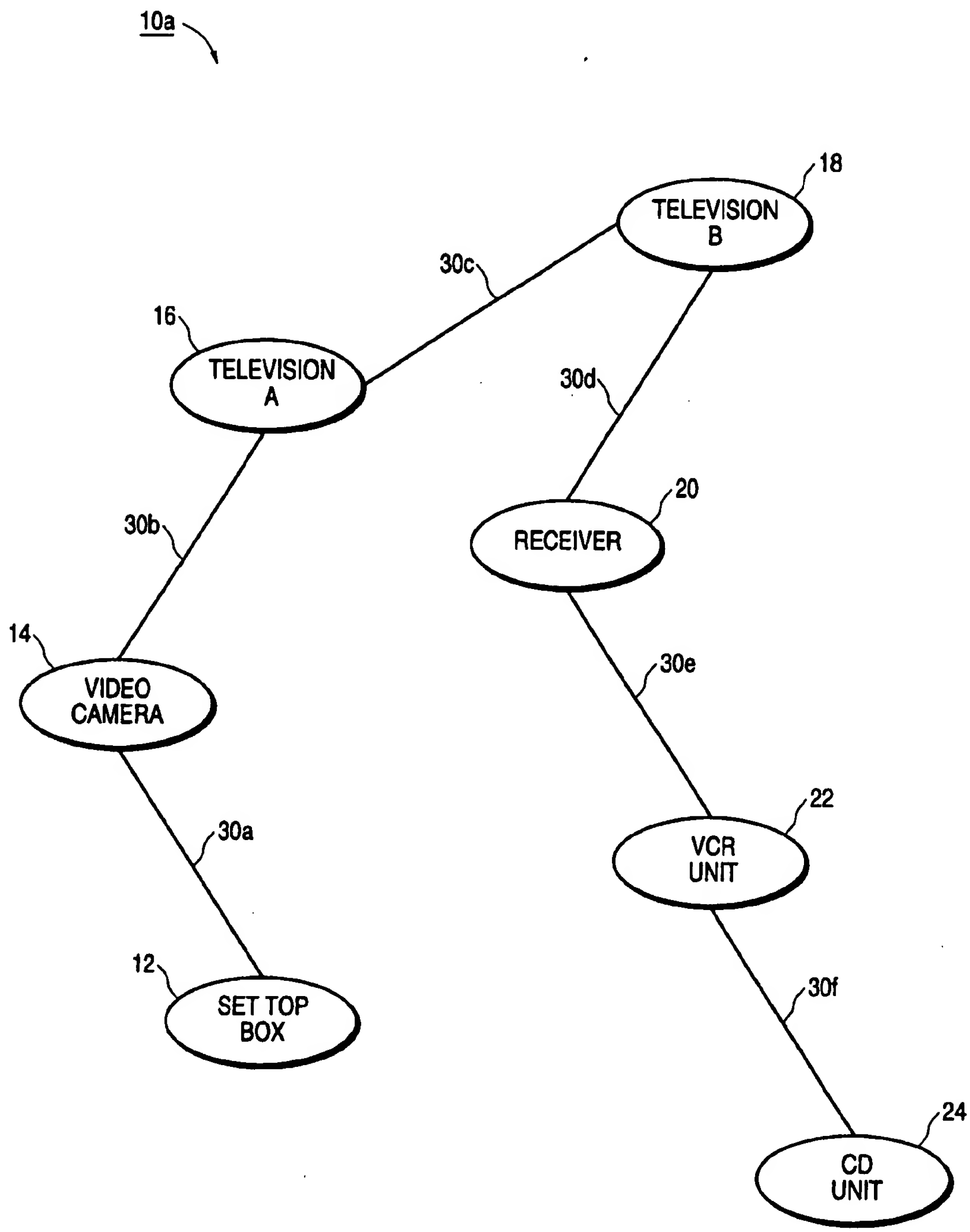

10a
18
TELEVISION B
30c
16
TELEVISION A
30d
20
RECEIVER
30b
14
VIDEO CAMERA
30e
22
VCR UNIT
30a
12
SET TOP BOX
30f
24
CD UNIT

FIG. 1A

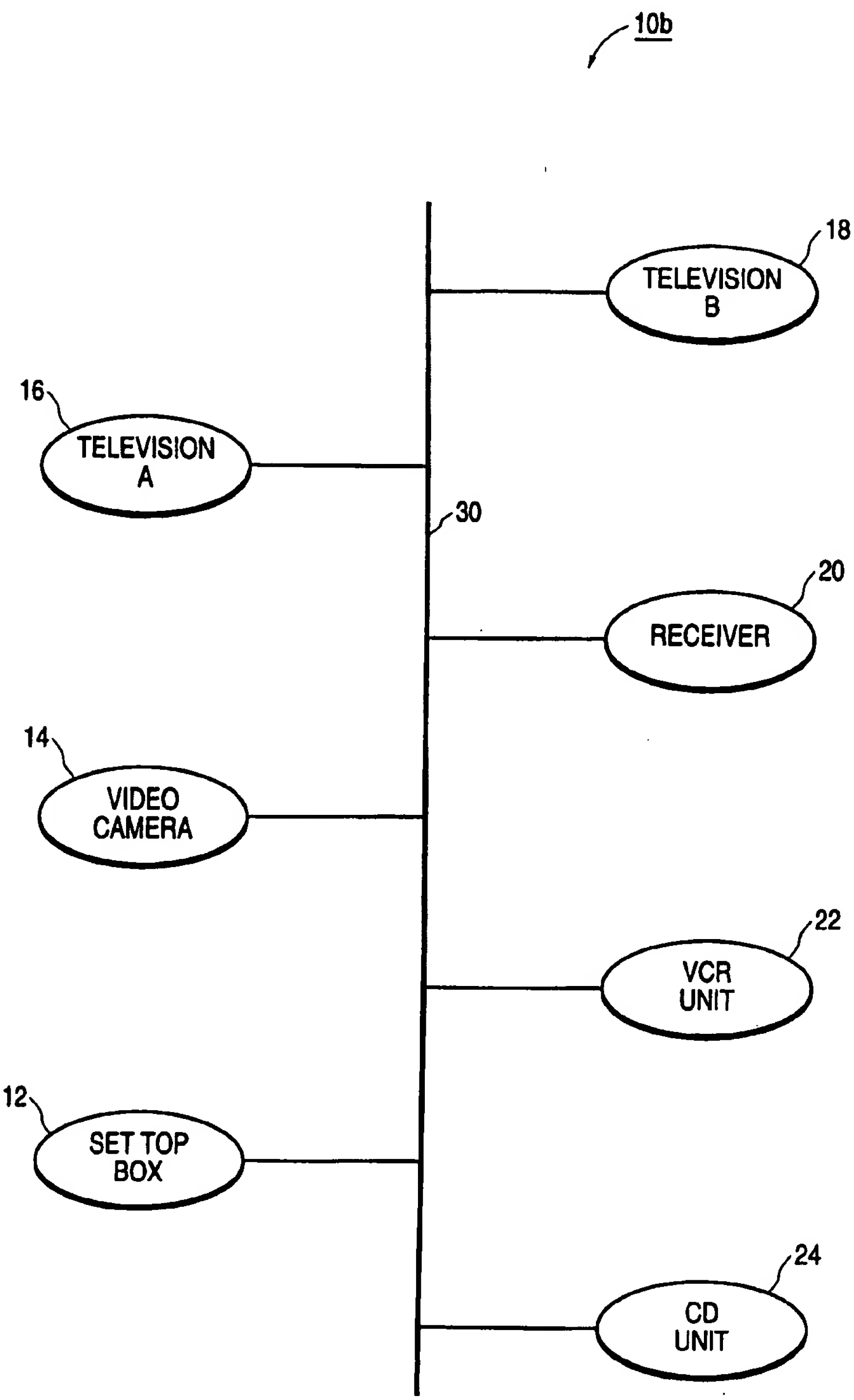

10b
18
TELEVISION B
16
TELEVISION A
30
20
RECEIVER
14
VIDEO CAMERA
22
VCR UNIT
12
SET TOP BOX
24
CD UNIT

FIG.1B

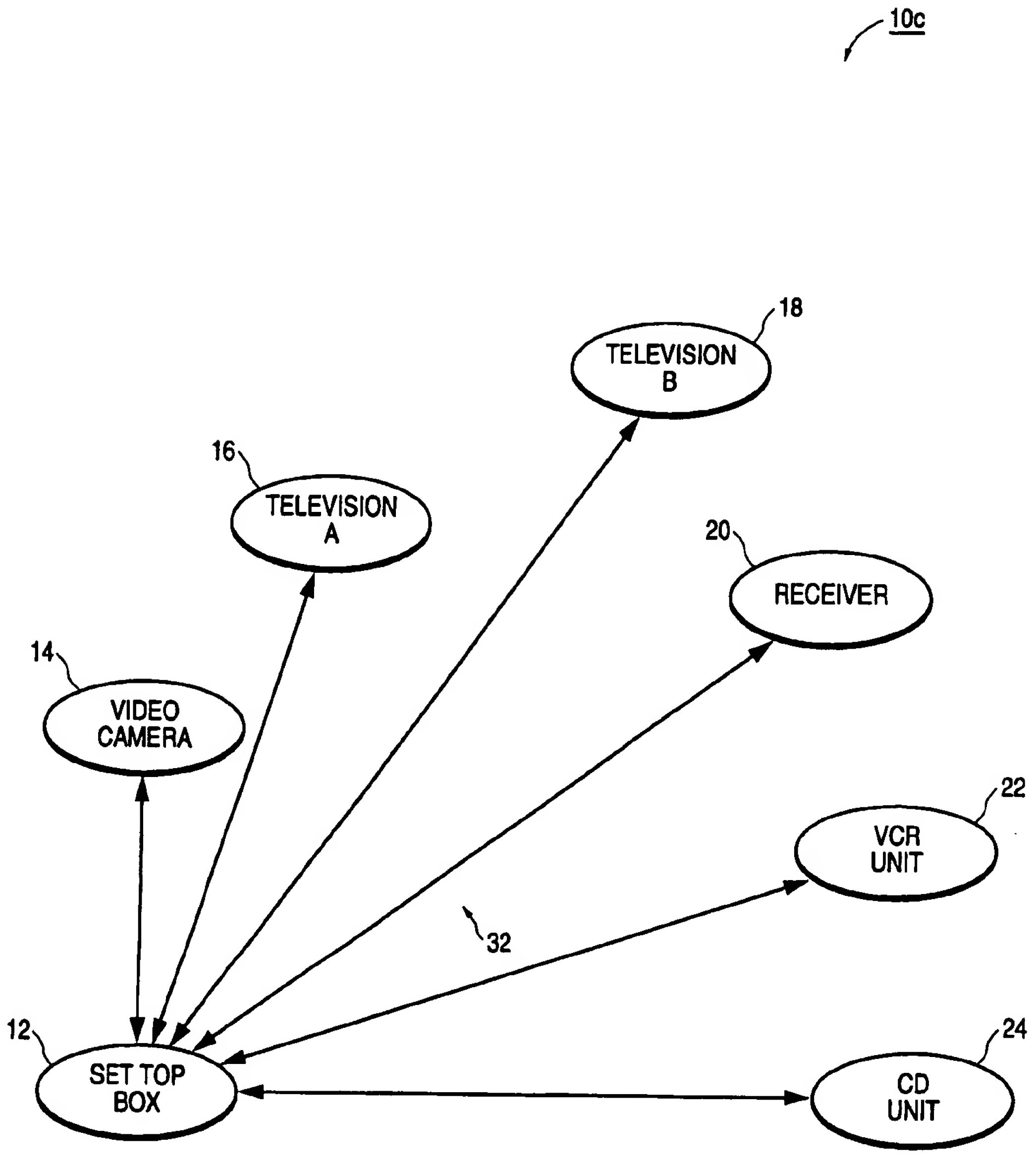

10c
18
TELEVISION B
16
TELEVISION A
20
RECEIVER
14
VIDEO CAMERA
22
VCR UNIT
32
12
SET TOP BOX
24
CD UNIT

FIG.1C

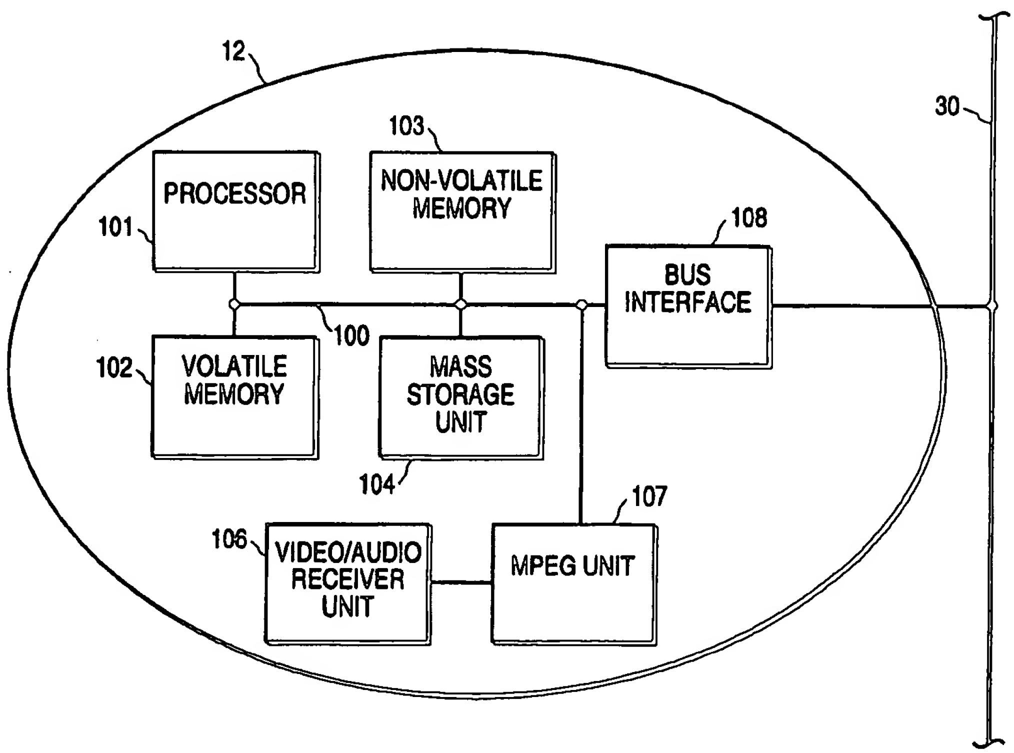

12
30
103
NON-VOLATILE MEMORY
PROCESSOR
101
108
BUS INTERFACE
100
102
VOLATILE MEMORY
MASS STORAGE UNIT
104
107
106
VIDEO/AUDIO RECEIVER UNIT
MPEG UNIT

FIG.2

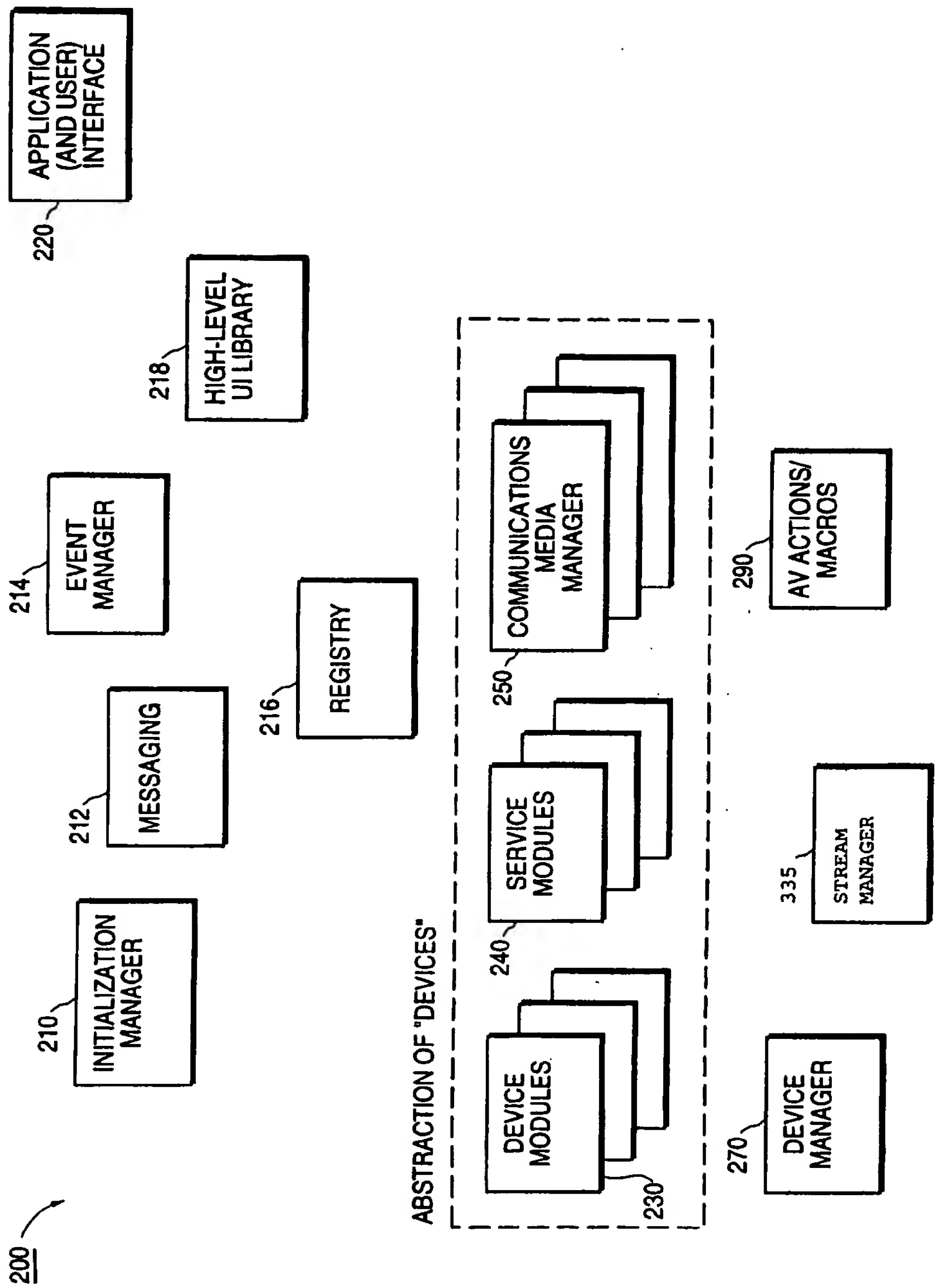

APPLICATION (AND USER) INTERFACE
220
HIGH-LEVEL UI LIBRARY
218
EVENT MANAGER
214
REGISTRY
216
MESSAGING
212
INITIALIZATION MANAGER
210
ABSTRACTION OF "DEVICES"
COMMUNICATIONS MEDIA MANAGER
250
SERVICE MODULES
240
DEVICE MODULES
230
AV ACTIONS/ MACROS
290
STREAM MANAGER
335
DEVICE MANAGER
270
200

FIG.3

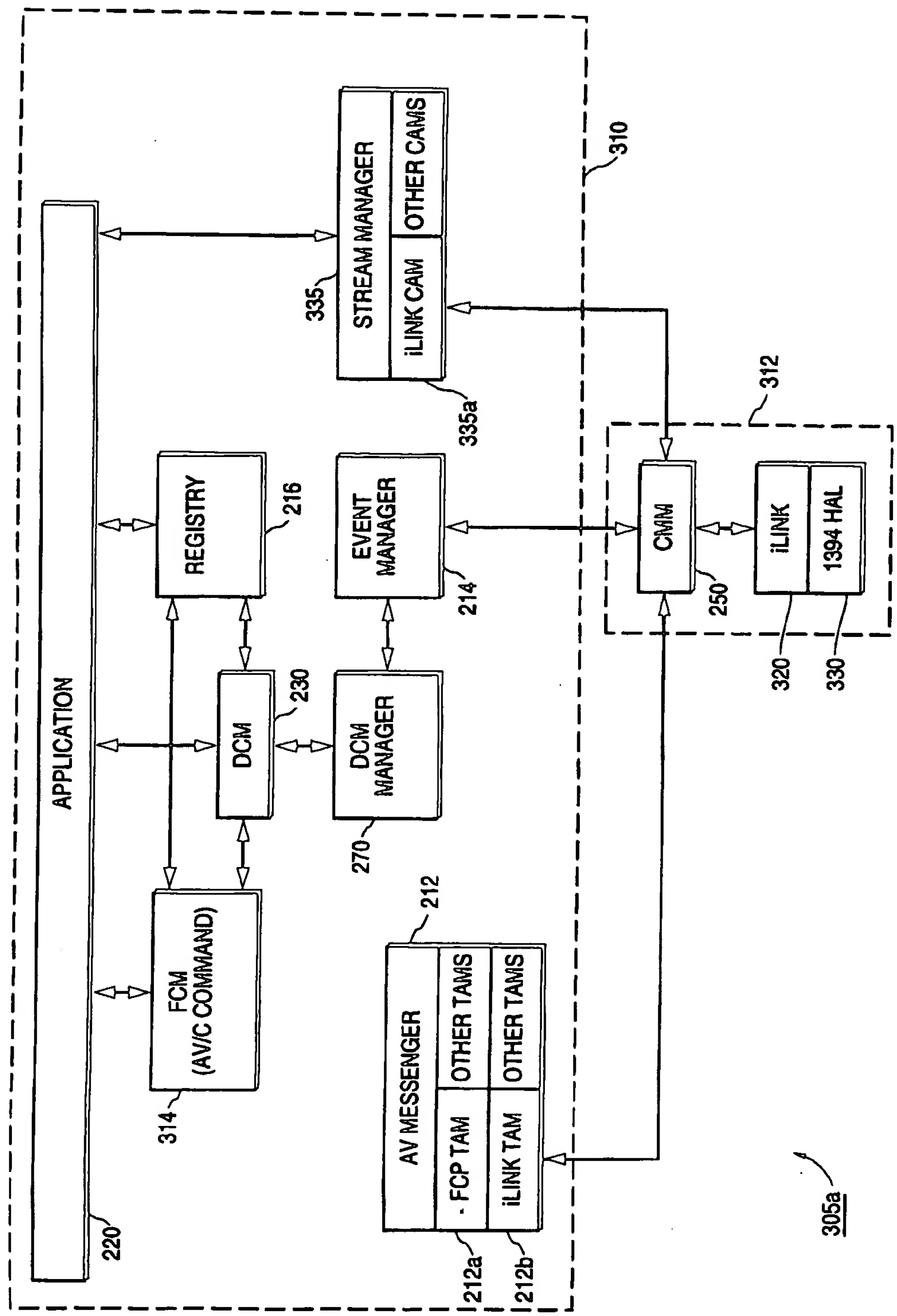

APPLICATION
220
STREAM MANAGER
iLINK CAM
OTHER CAMS
335
335a
310
REGISTRY
216
EVENT MANAGER
214
DCM
230
DCM MANAGER
270
FCM (AV/C COMMAND)
314
AV MESSENGER
FCP TAM
iLINK TAM
OTHER TAMS
OTHER TAMS
212
212a
212b
CMM
250
iLINK
1394 HAL
312
320
330
305a

FIG. 4

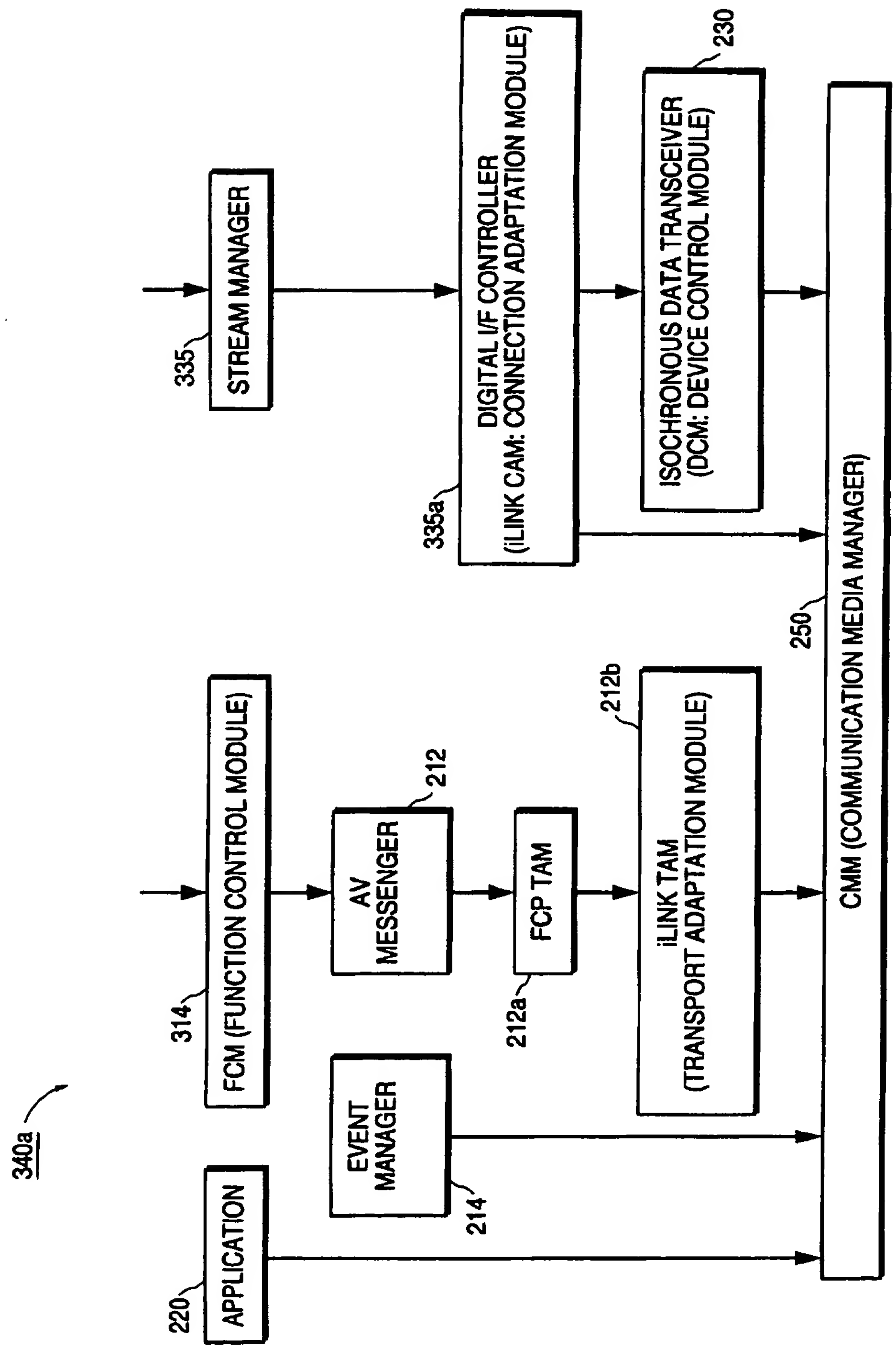

340a
APPLICATION
220
EVENT MANAGER
214
FCM (FUNCTION CONTROL MODULE)
314
AV MESSENGER
212
FCP TAM
212a
iLINK TAM (TRANSPORT ADAPTATION MODULE)
212b
STREAM MANAGER
335
DIGITAL I/F CONTROLLER (iLINK CAM: CONNECTION ADAPTATION MODULE)
335a
ISOCHRONOUS DATA TRANSCEIVER (DCM: DEVICE CONTROL MODULE)
230
CMM (COMMUNICATION MEDIA MANAGER)
250

FIG.5A

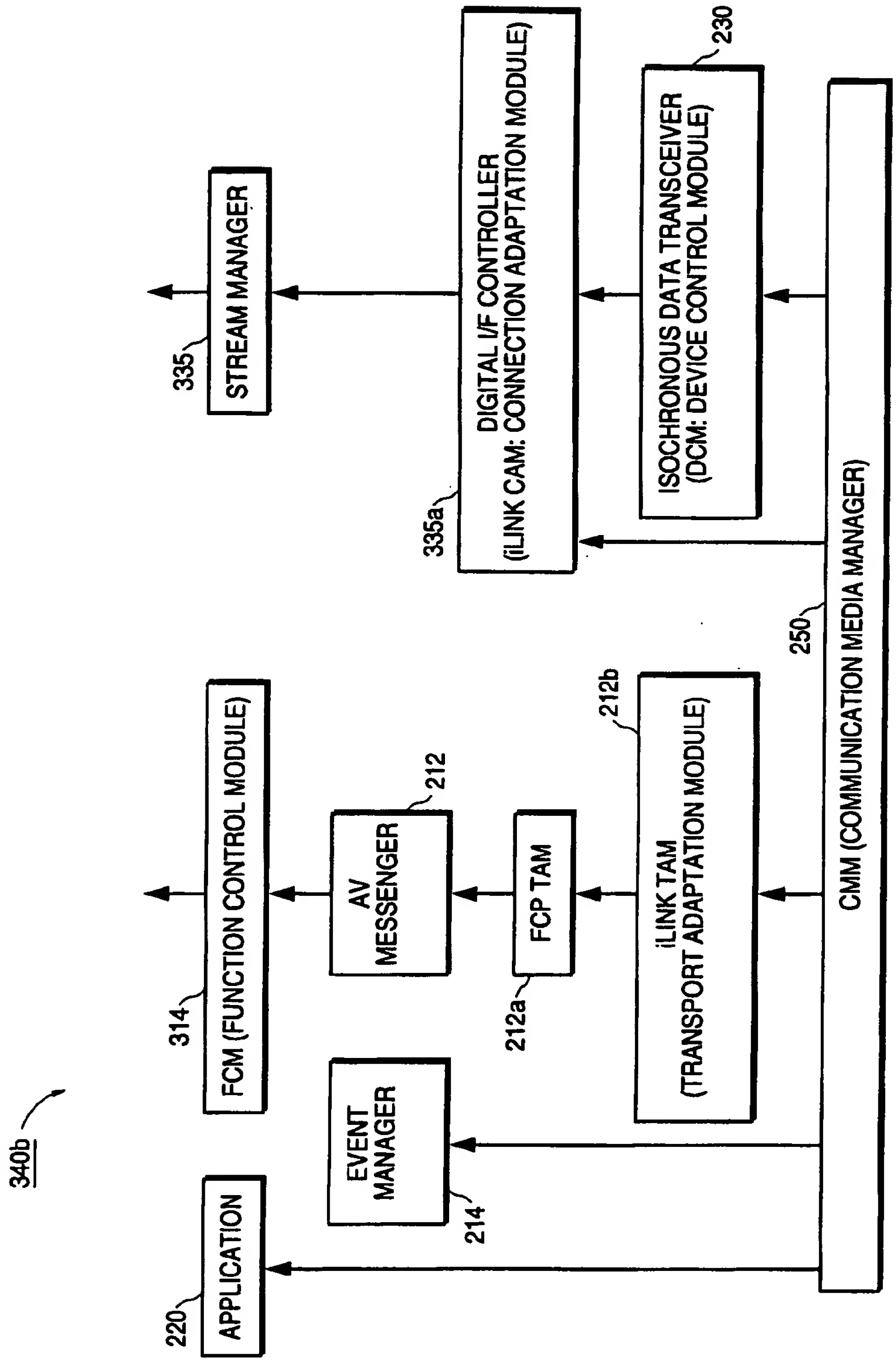

340b
APPLICATION
220
EVENT MANAGER
214
FCM (FUNCTION CONTROL MODULE)
314
AV MESSENGER
212
FCP TAM
212a
iLINK TAM (TRANSPORT ADAPTATION MODULE)
212b
CMM (COMMUNICATION MEDIA MANAGER)
250
STREAM MANAGER
335
DIGITAL I/F CONTROLLER (iLINK CAM: CONNECTION ADAPTATION MODULE)
335a
ISOCHRONOUS DATA TRANSCEIVER (DCM: DEVICE CONTROL MODULE)
230

FIG.5B

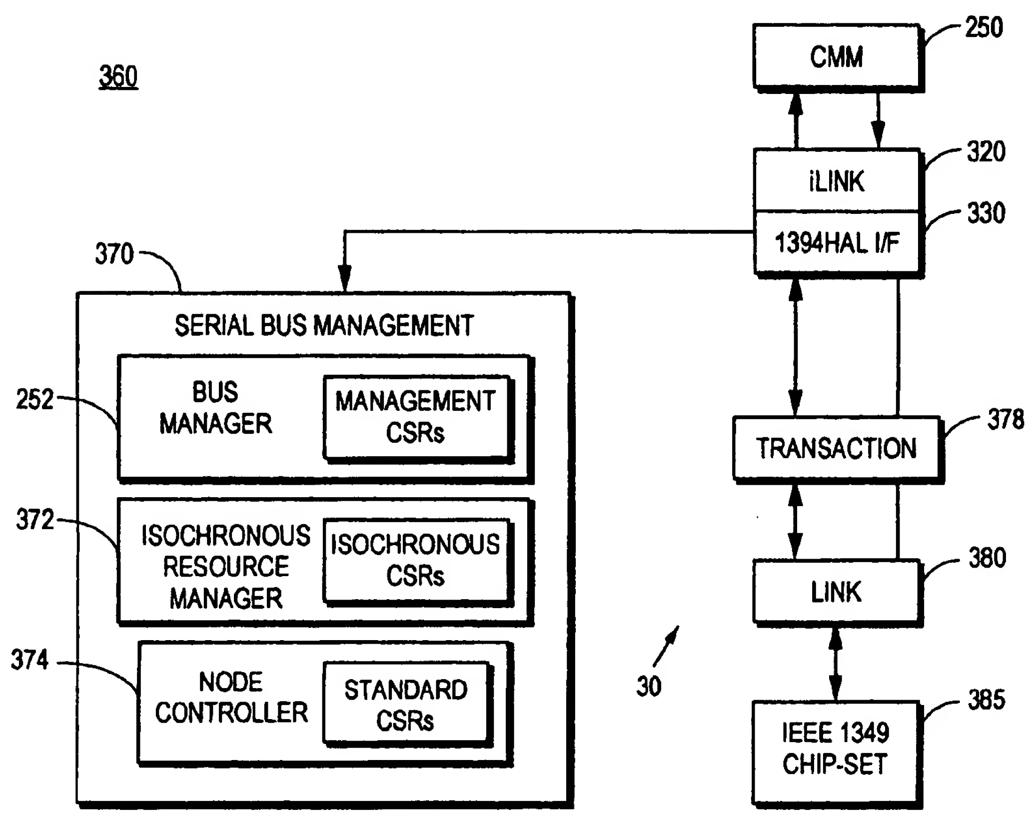

360
CMM
250
iLINK
320
1394HAL I/F
330
370
SERIAL BUS MANAGEMENT
252
BUS MANAGER
MANAGEMENT CSRs
372
ISOCHRONOUS RESOURCE MANAGER
ISOCHRONOUS CSRs
374
NODE CONTROLLER
STANDARD CSRs
TRANSACTION
378
LINK
380
30
IEEE 1349 CHIP-SET
385

FIG. 6

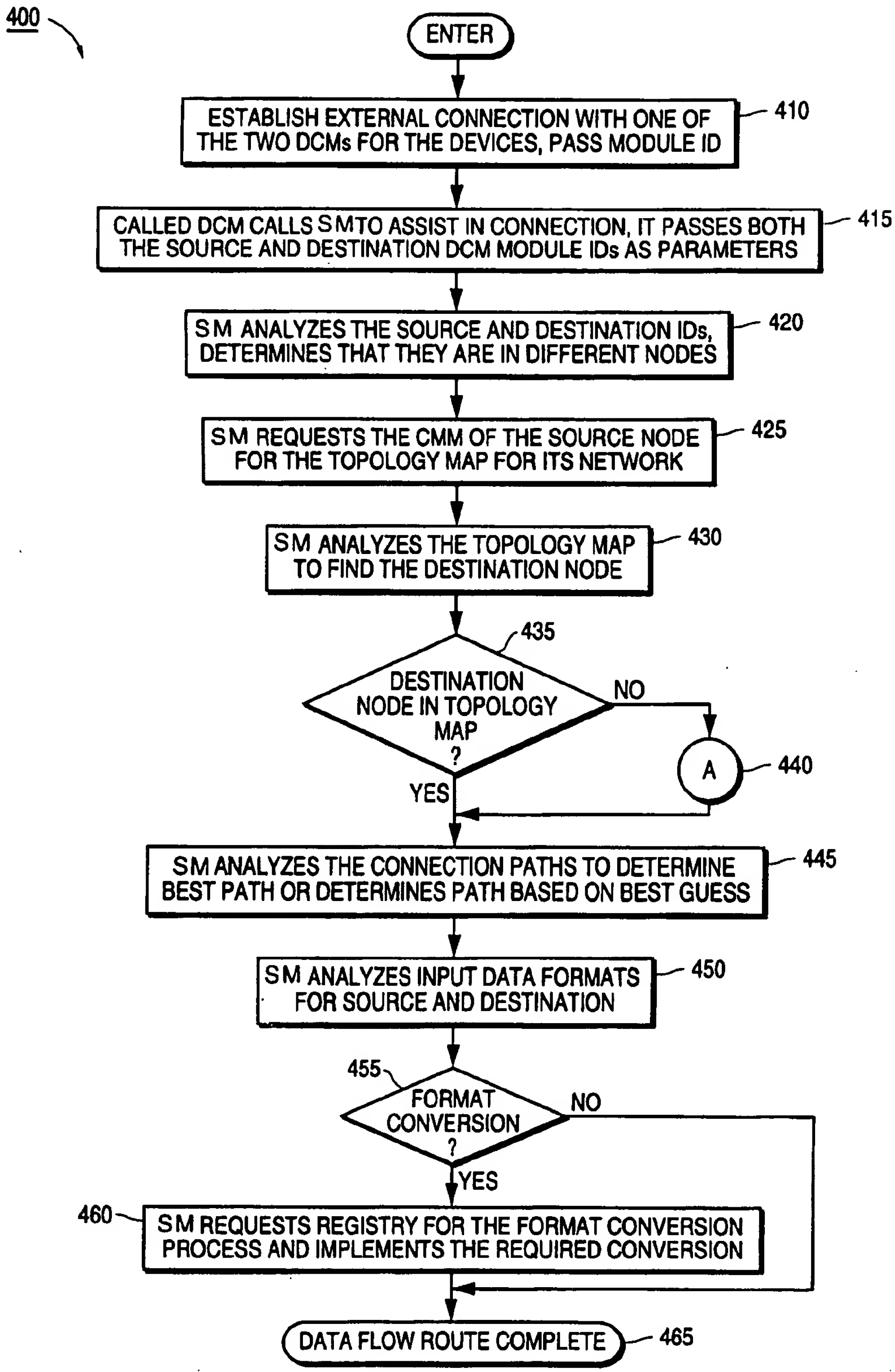

400
ENTER
ESTABLISH EXTERNAL CONNECTION WITH ONE OF THE TWO DCMs FOR THE DEVICES, PASS MODULE ID
410
CALLED DCM CALLS SM TO ASSIST IN CONNECTION, IT PASSES BOTH THE SOURCE AND DESTINATION DCM MODULE IDs AS PARAMETERS
415
SM ANALYZES THE SOURCE AND DESTINATION IDs, DETERMINES THAT THEY ARE IN DIFFERENT NODES
420
SM REQUESTS THE CMM OF THE SOURCE NODE FOR THE TOPOLOGY MAP FOR ITS NETWORK
425
SM ANALYZES THE TOPOLOGY MAP TO FIND THE DESTINATION NODE
430
435
DESTINATION NODE IN TOPOLOGY MAP ?
NO
A
440
YES
SM ANALYZES THE CONNECTION PATHS TO DETERMINE BEST PATH OR DETERMINES PATH BASED ON BEST GUESS
445
SM ANALYZES INPUT DATA FORMATS FOR SOURCE AND DESTINATION
450
455
FORMAT CONVERSION ?
NO
YES
460
SM REQUESTS REGISTRY FOR THE FORMAT CONVERSION PROCESS AND IMPLEMENTS THE REQUIRED CONVERSION
DATA FLOW ROUTE COMPLETE
465

FIG.7A

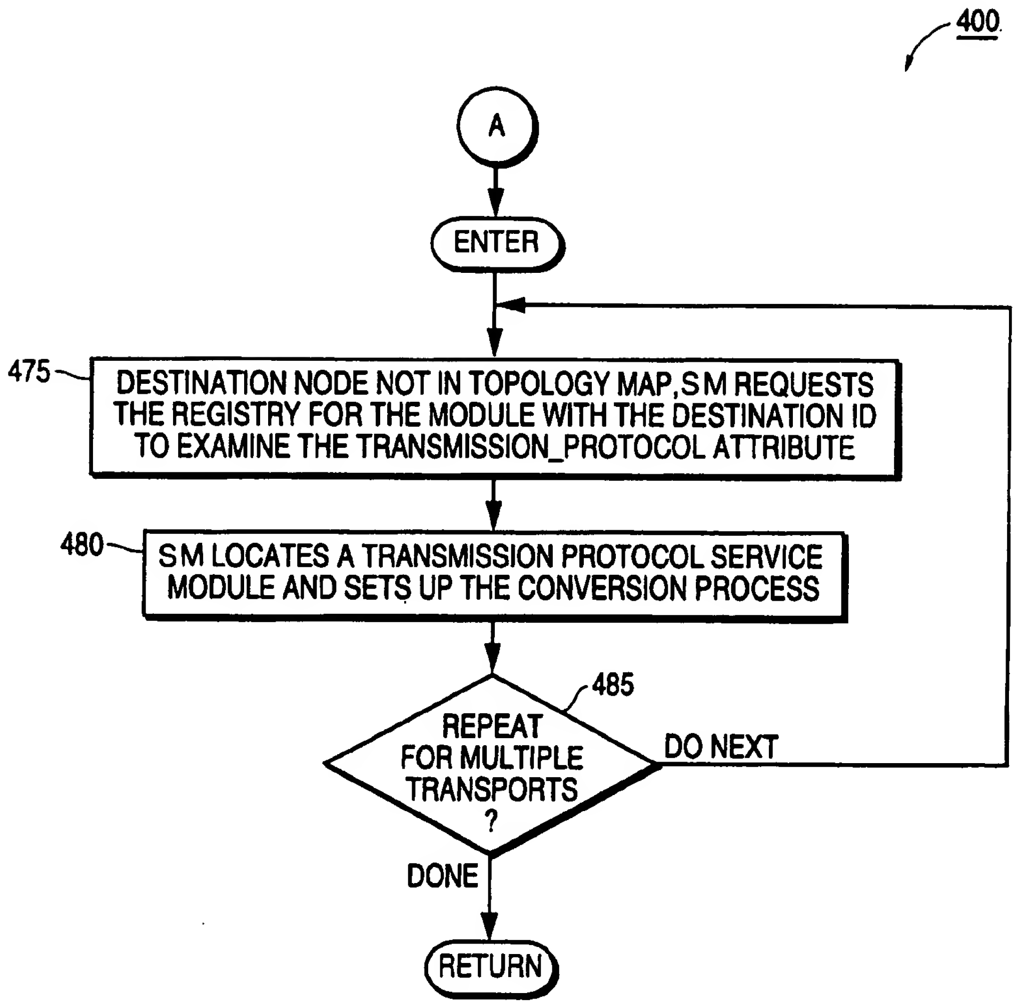

400
A
ENTER
475
DESTINATION NODE NOT IN TOPOLOGY MAP,SM REQUESTS THE REGISTRY FOR THE MODULE WITH THE DESTINATION ID TO EXAMINE THE TRANSMISSION_PROTOCOL ATTRIBUTE
480
SM LOCATES A TRANSMISSION PROTOCOL SERVICE MODULE AND SETS UP THE CONVERSION PROCESS
485
REPEAT FOR MULTIPLE TRANSPORTS ?
DO NEXT
DONE
RETURN

FIG.7B

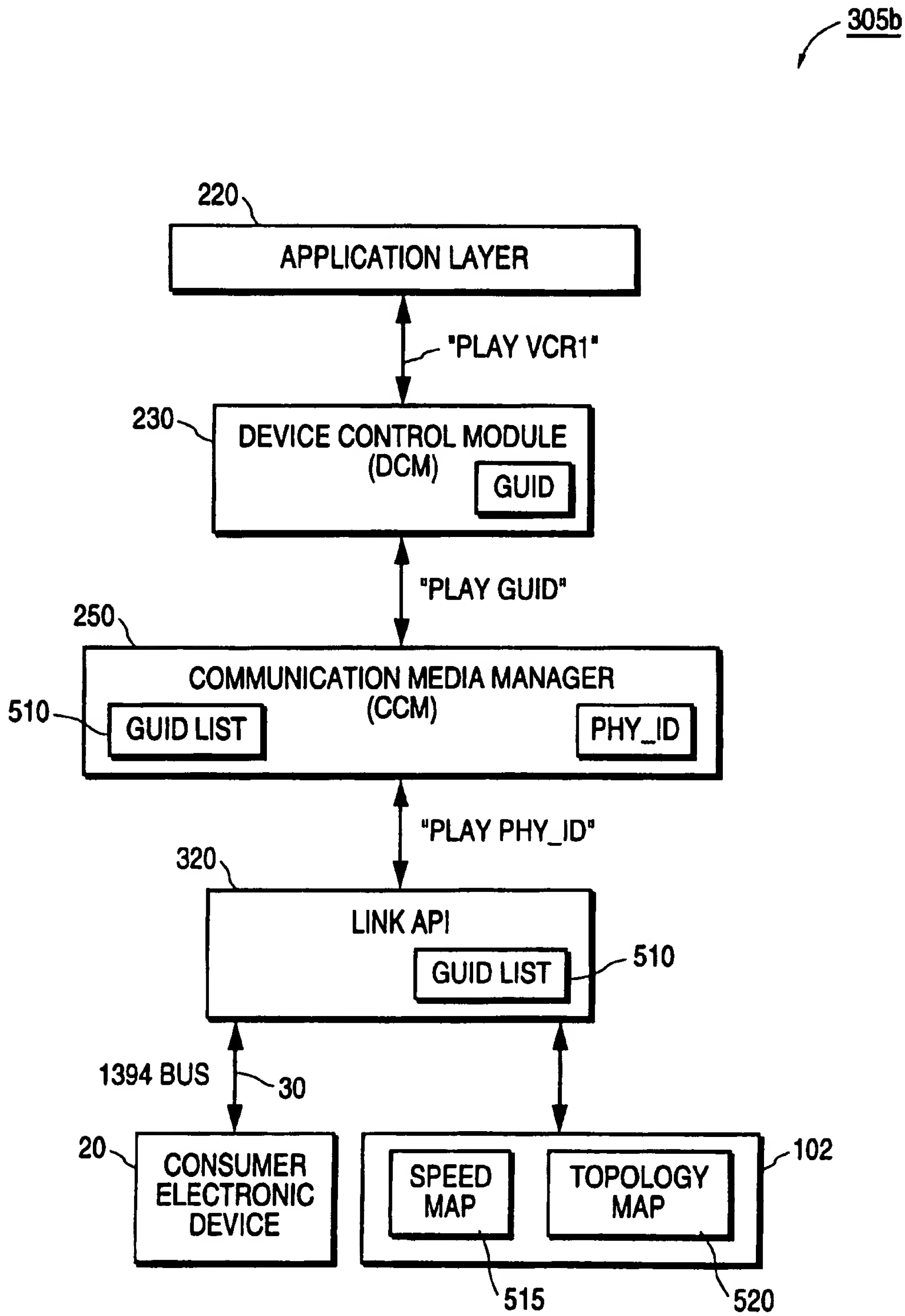

305b
220
APPLICATION LAYER
"PLAY VCR1"
230
DEVICE CONTROL MODULE (DCM)
GUID
"PLAY GUID"
250
COMMUNICATION MEDIA MANAGER (CCM)
510
GUID LIST
PHY_ID
"PLAY PHY_ID"
320
LINK API
GUID LIST
510
1394 BUS
30
20
CONSUMER ELECTRONIC DEVICE
SPEED MAP
TOPOLOGY MAP
102
515
520

FIG.8

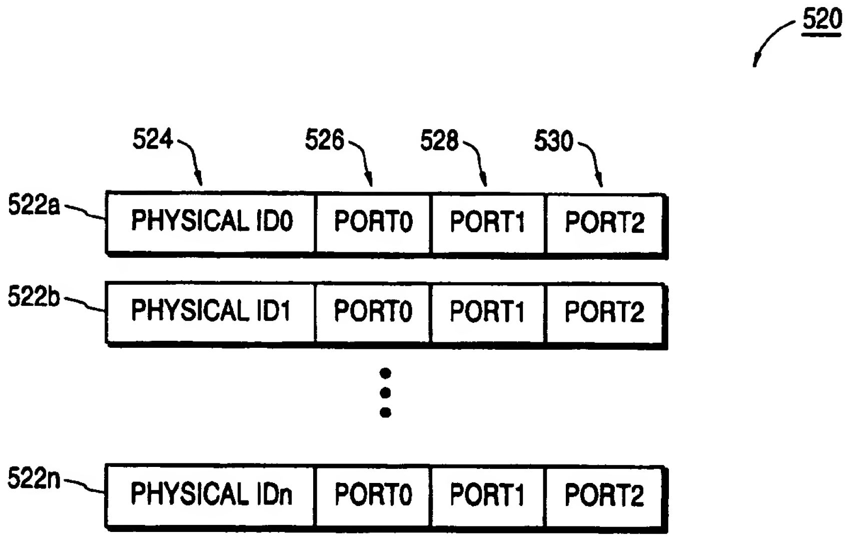

520
524
526
528
530
522a
PHYSICAL ID0
PORT0
PORT1
PORT2
522b
PHYSICAL ID1
PORT0
PORT1
PORT2
522n
PHYSICAL IDn
PORT0
PORT1
PORT2

FIG.9A

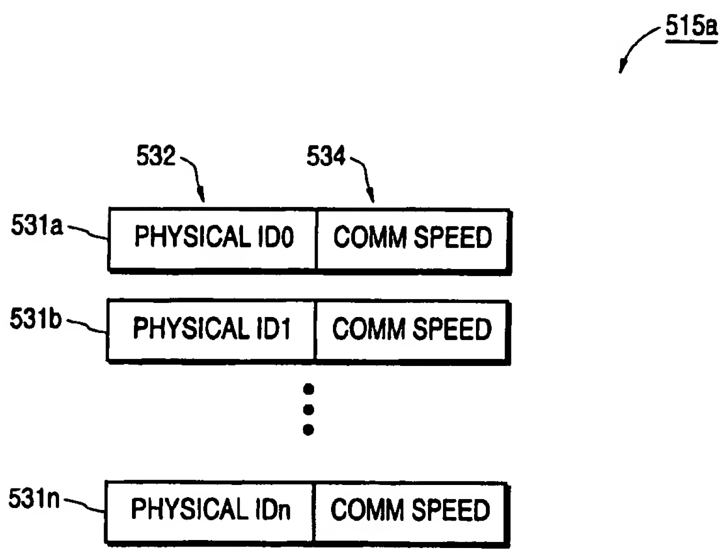

515a
532
534
531a
PHYSICAL ID0
COMM SPEED
531b
PHYSICAL ID1
COMM SPEED
531n
PHYSICAL IDn
COMM SPEED

FIG.9B

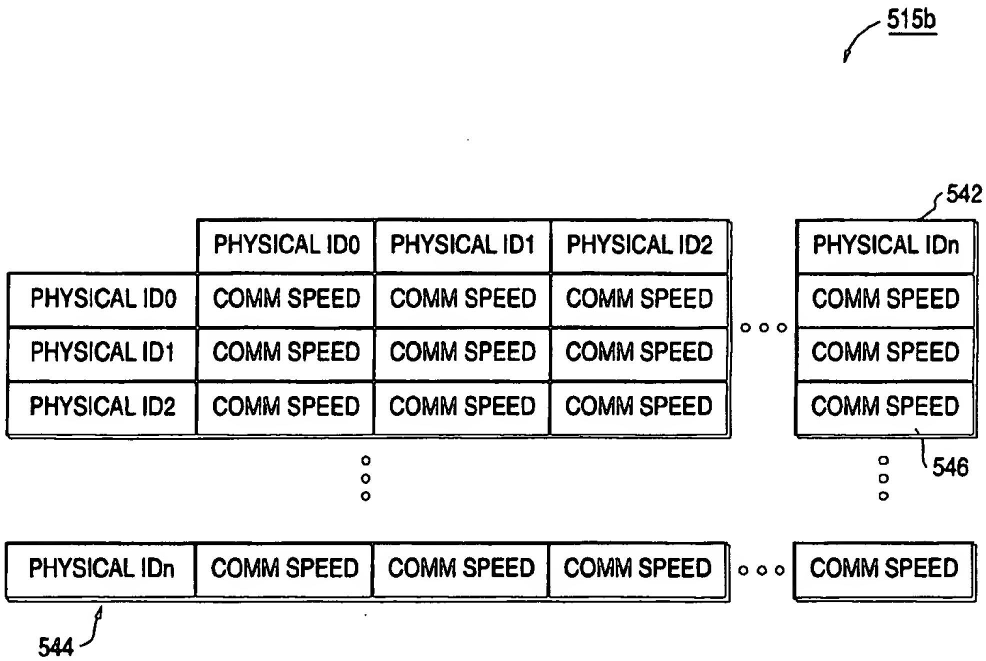

515b
542
PHYSICAL ID0
PHYSICAL ID1
PHYSICAL ID2
PHYSICAL IDn
PHYSICAL ID0
PHYSICAL ID1
PHYSICAL ID2
COMM SPEED
546
PHYSICAL IDn
COMM SPEED
544

FIG. 9C

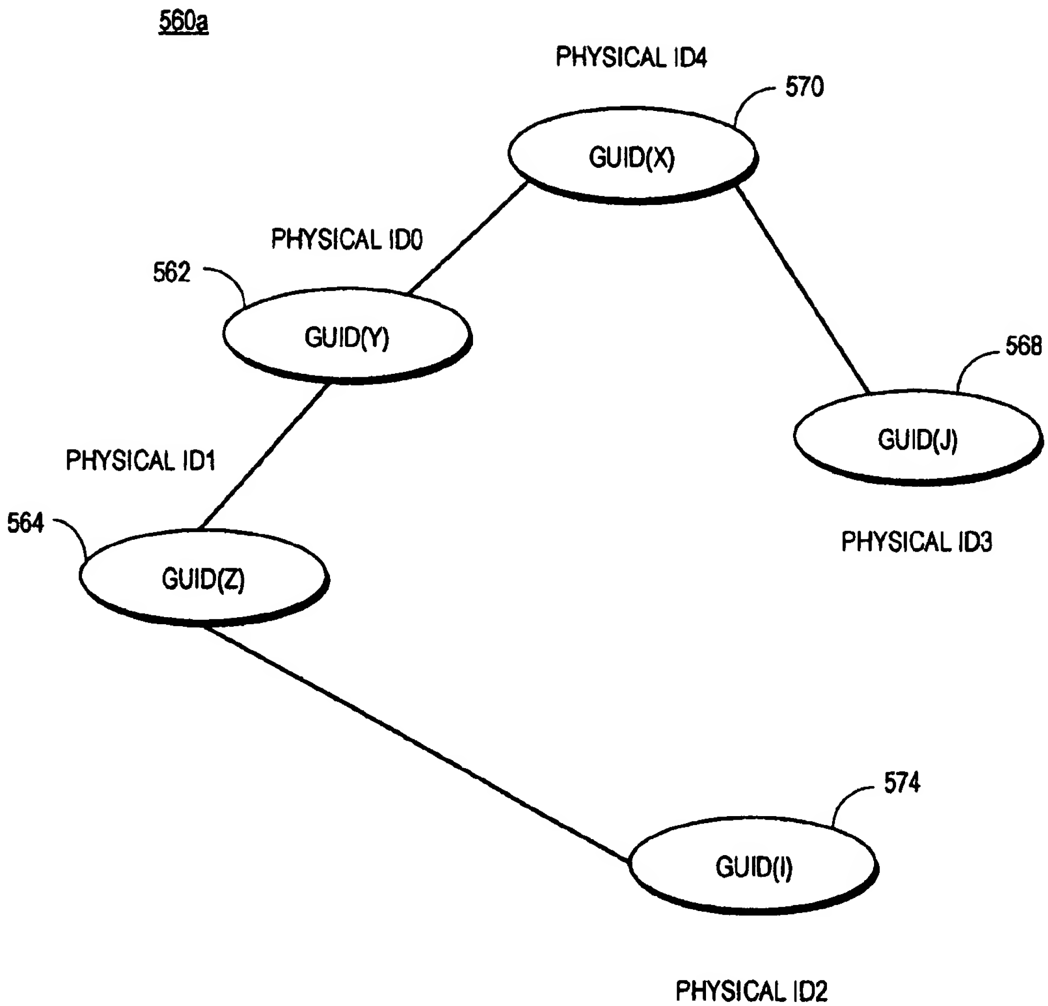
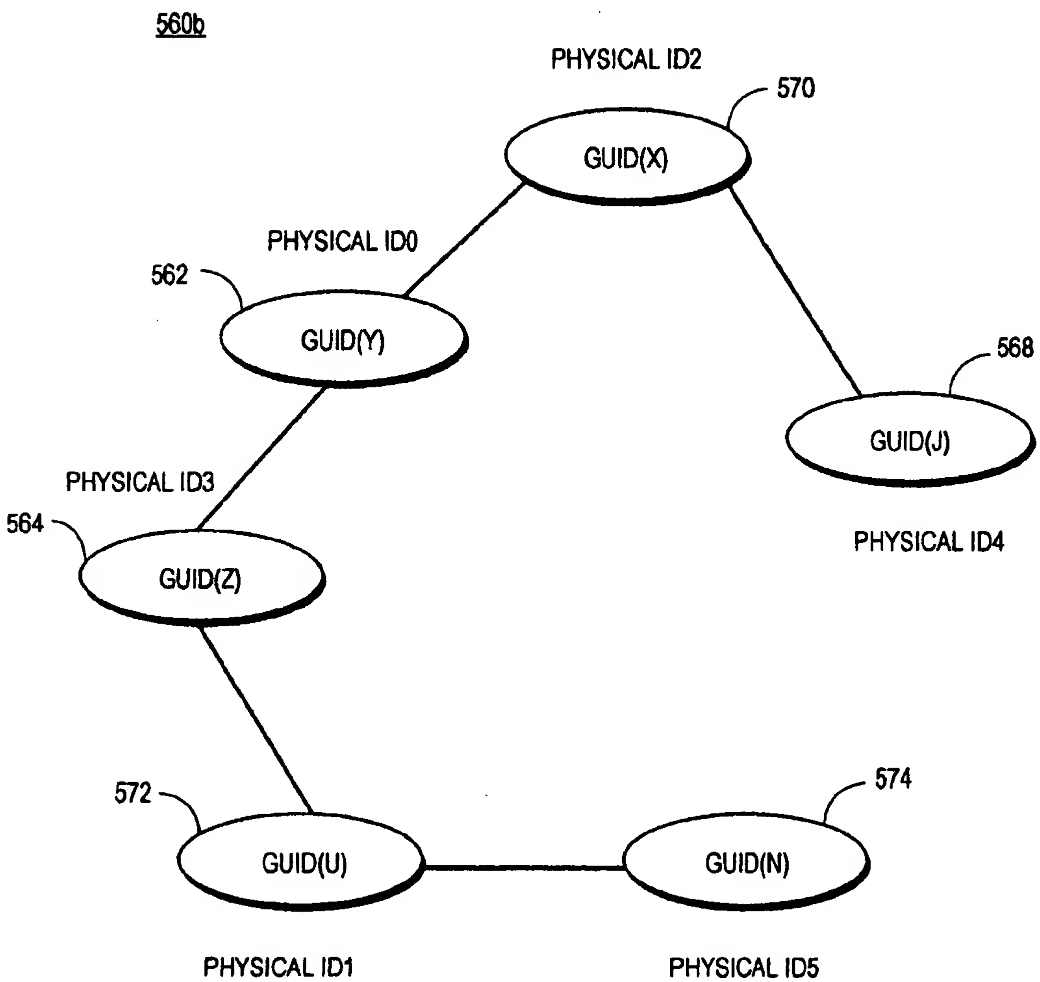

560a
PHYSICAL ID4
570
GUID(X)
PHYSICAL ID0
562
GUID(Y)
568
GUID(J)
PHYSICAL ID1
564
GUID(Z)
PHYSICAL ID3
574
GUID(I)
PHYSICAL ID2

FIG. 10A



560b
PHYSICAL ID2
570
GUID(X)
PHYSICAL ID0
562
GUID(Y)
568
GUID(J)
PHYSICAL ID3
564
GUID(Z)
PHYSICAL ID4
572
GUID(U)
574
GUID(N)
PHYSICAL ID1
PHYSICAL ID5

FIG. 10B

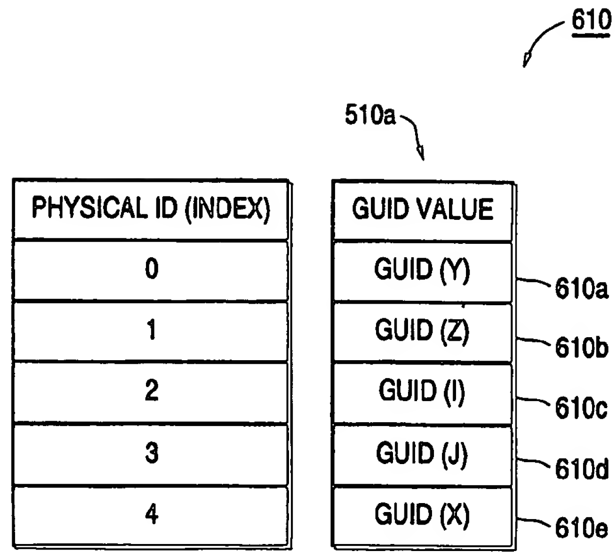


FIG. 11A

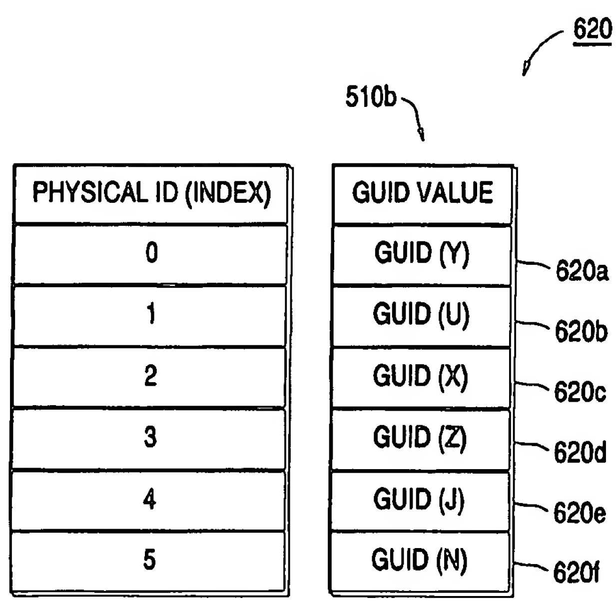


FIG. 11B

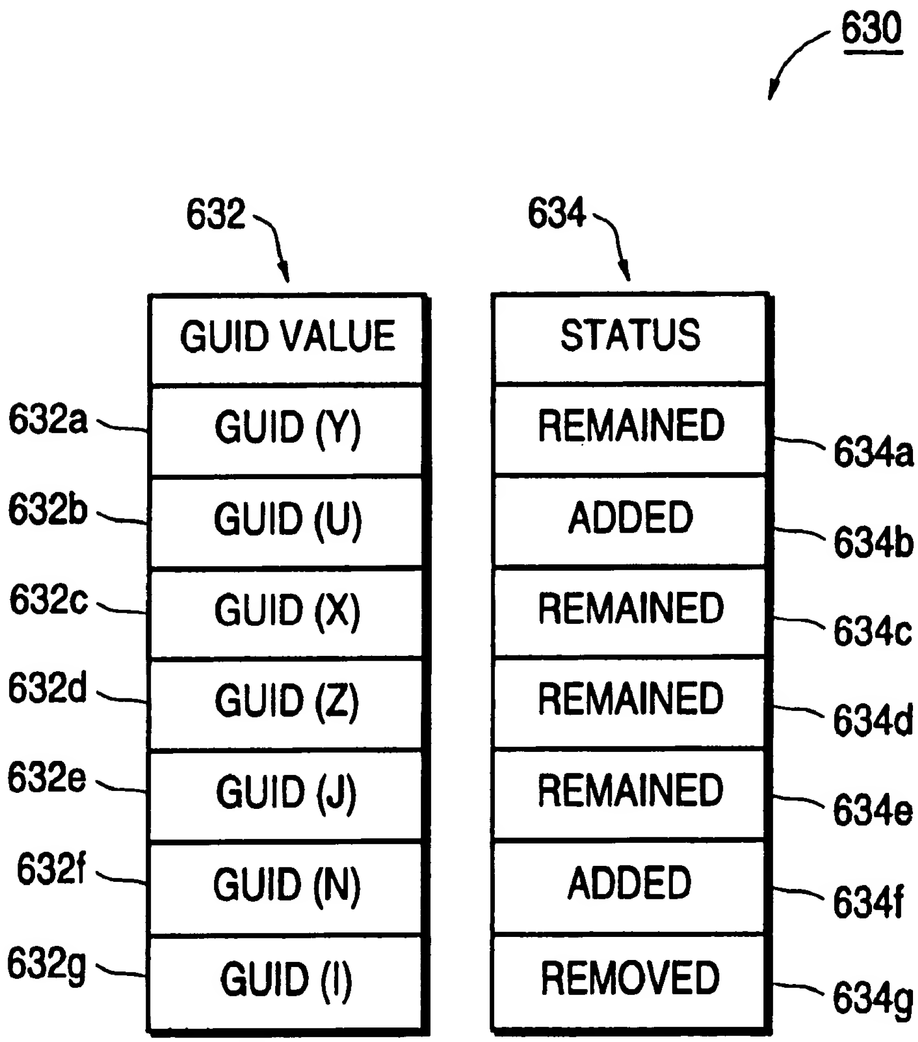

630
632
634
GUID VALUE
STATUS
632a
GUID (Y)
REMAINED
634a
632b
GUID (U)
ADDED
634b
632c
GUID (X)
REMAINED
634c
632d
GUID (Z)
REMAINED
634d
632e
GUID (J)
REMAINED
634e
632f
GUID (N)
ADDED
634f
632g
GUID (I)
REMOVED
634g

FIG.12A

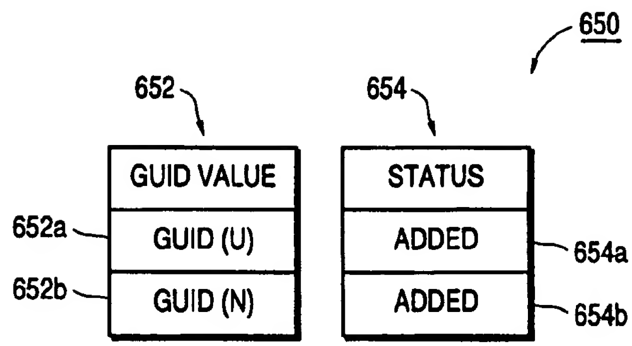


FIG. 12B

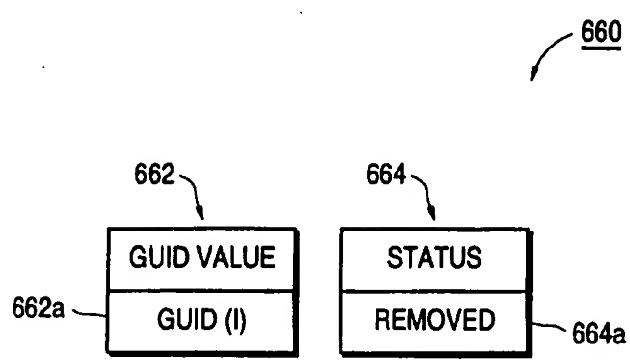


FIG. 12C

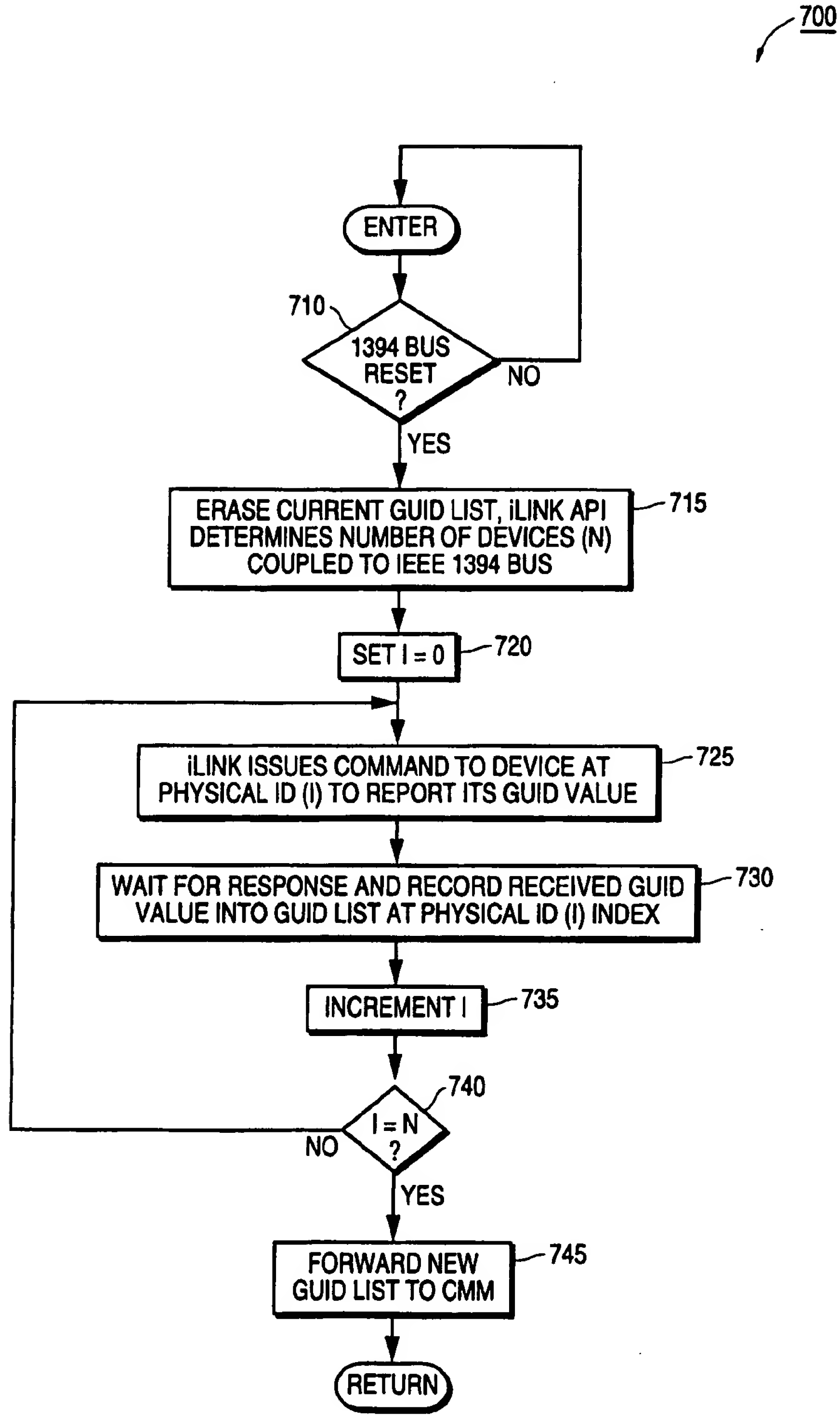

700
ENTER
710
1394 BUS RESET ?
NO
YES
ERASE CURRENT GUID LIST, iLINK API DETERMINES NUMBER OF DEVICES (N) COUPLED TO IEEE 1394 BUS
715
SET I = 0
720
iLINK ISSUES COMMAND TO DEVICE AT PHYSICAL ID (I) TO REPORT ITS GUID VALUE
725
WAIT FOR RESPONSE AND RECORD RECEIVED GUID VALUE INTO GUID LIST AT PHYSICAL ID (I) INDEX
730
INCREMENT I
735
I = N ?
740
NO
YES
FORWARD NEW GUID LIST TO CMM
745
RETURN

FIG.13

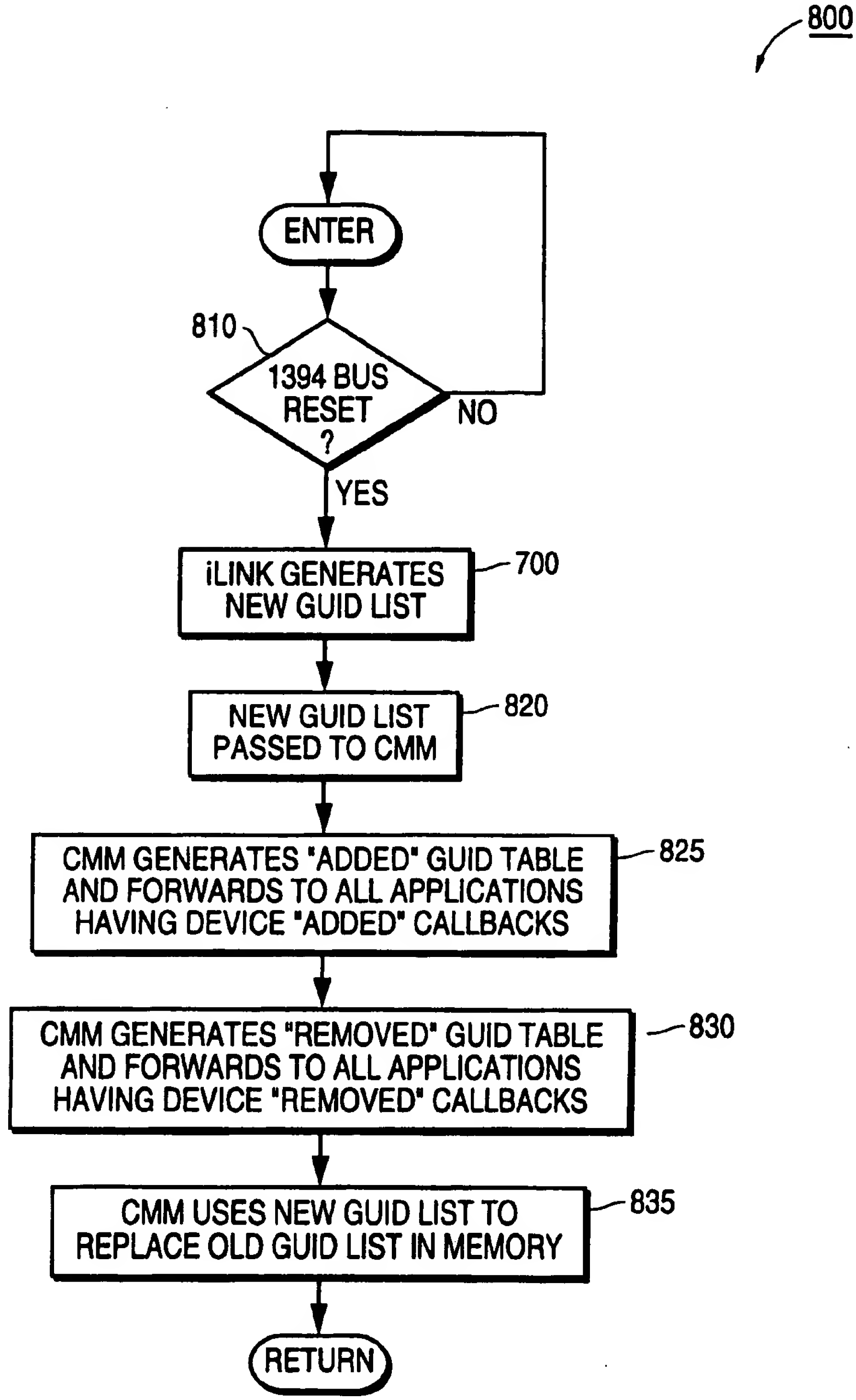

800
ENTER
810
1394 BUS RESET ?
NO
YES
iLINK GENERATES NEW GUID LIST
700
NEW GUID LIST PASSED TO CMM
820
CMM GENERATES "ADDED" GUID TABLE AND FORWARDS TO ALL APPLICATIONS HAVING DEVICE "ADDED" CALLBACKS
825
CMM GENERATES "REMOVED" GUID TABLE AND FORWARDS TO ALL APPLICATIONS HAVING DEVICE "REMOVED" CALLBACKS
830
CMM USES NEW GUID LIST TO REPLACE OLD GUID LIST IN MEMORY
835
RETURN

FIG.14

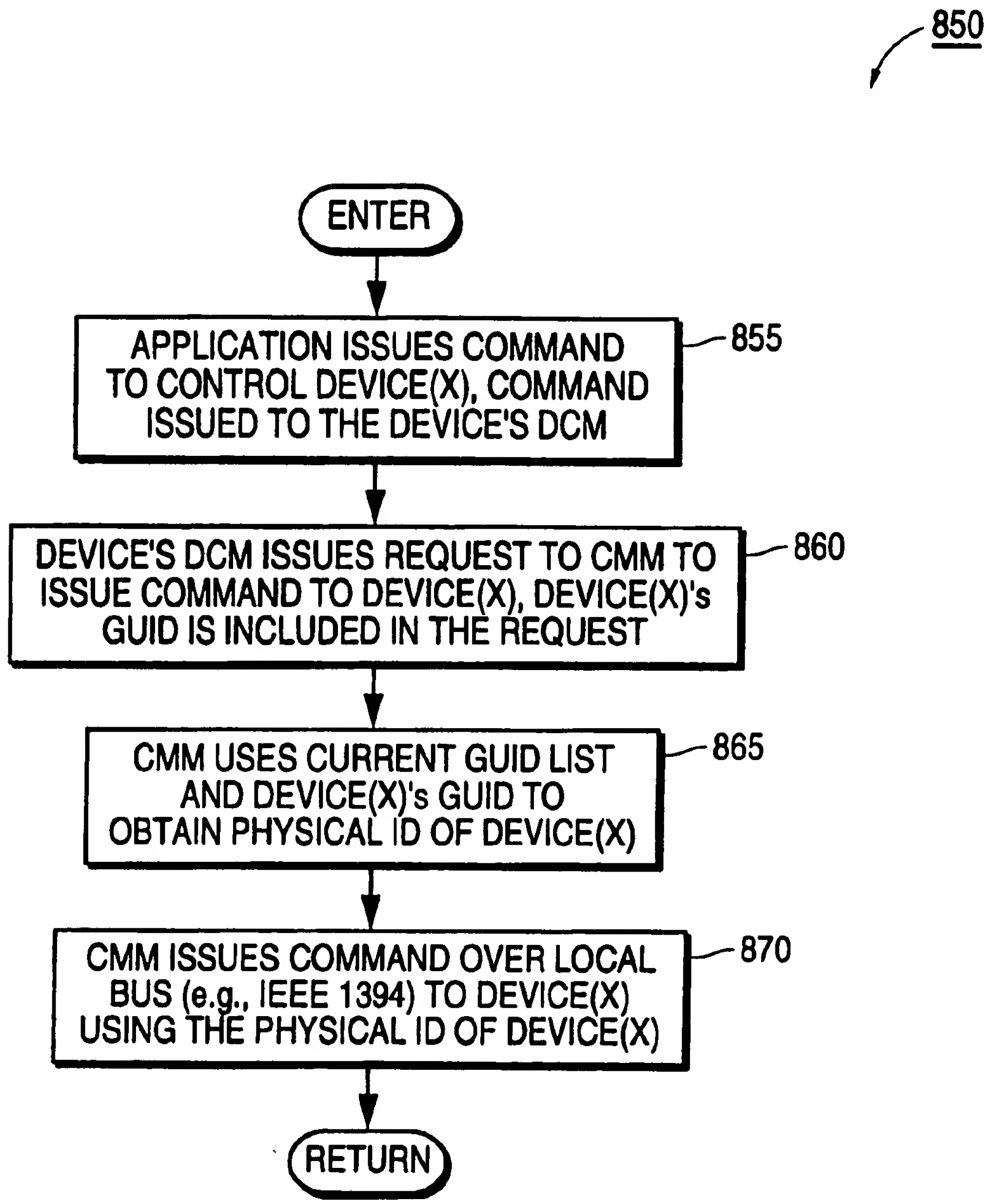

850
ENTER
APPLICATION ISSUES COMMAND TO CONTROL DEVICE(X), COMMAND ISSUED TO THE DEVICE'S DCM
855
DEVICE'S DCM ISSUES REQUEST TO CMM TO ISSUE COMMAND TO DEVICE(X), DEVICE(X)'s GUID IS INCLUDED IN THE REQUEST
860
CMM USES CURRENT GUID LIST AND DEVICE(X)'s GUID TO OBTAIN PHYSICAL ID OF DEVICE(X)
865
CMM ISSUES COMMAND OVER LOCAL BUS (e.g., IEEE 1394) TO DEVICE(X) USING THE PHYSICAL ID OF DEVICE(X)
870
RETURN

FIG.15A

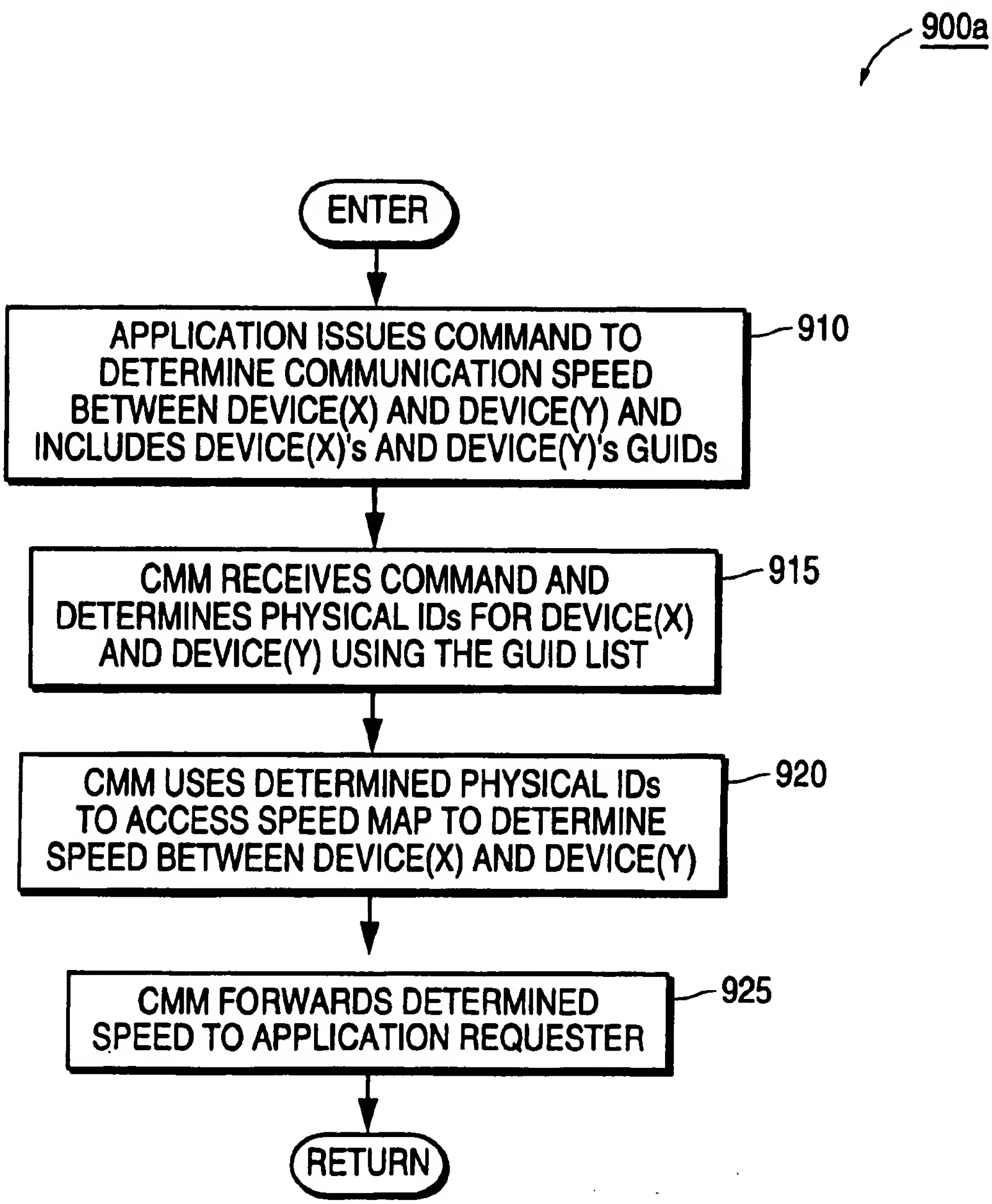

900a
ENTER
APPLICATION ISSUES COMMAND TO DETERMINE COMMUNICATION SPEED BETWEEN DEVICE(X) AND DEVICE(Y) AND INCLUDES DEVICE(X)'s AND DEVICE(Y)'s GUIDs
910
CMM RECEIVES COMMAND AND DETERMINES PHYSICAL IDs FOR DEVICE(X) AND DEVICE(Y) USING THE GUID LIST
915
CMM USES DETERMINED PHYSICAL IDs TO ACCESS SPEED MAP TO DETERMINE SPEED BETWEEN DEVICE(X) AND DEVICE(Y)
920
CMM FORWARDS DETERMINED SPEED TO APPLICATION REQUESTER
925
RETURN

FIG.15B

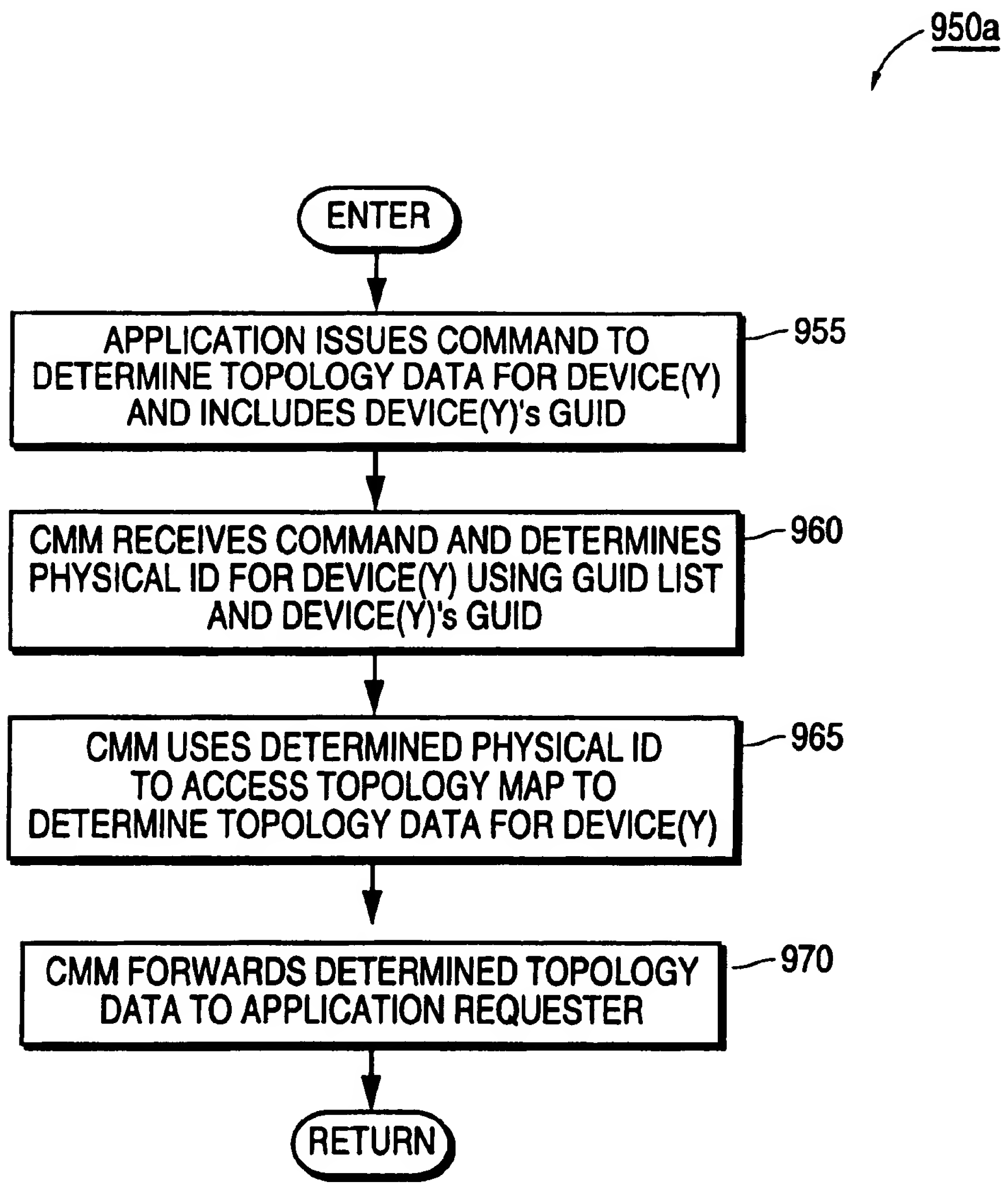

950a
ENTER
APPLICATION ISSUES COMMAND TO DETERMINE TOPOLOGY DATA FOR DEVICE(Y) AND INCLUDES DEVICE(Y)'s GUID
955
CMM RECEIVES COMMAND AND DETERMINES PHYSICAL ID FOR DEVICE(Y) USING GUID LIST AND DEVICE(Y)'s GUID
960
CMM USES DETERMINED PHYSICAL ID TO ACCESS TOPOLOGY MAP TO DETERMINE TOPOLOGY DATA FOR DEVICE(Y)
965
CMM FORWARDS DETERMINED TOPOLOGY DATA TO APPLICATION REQUESTER
970
RETURN

FIG.15C

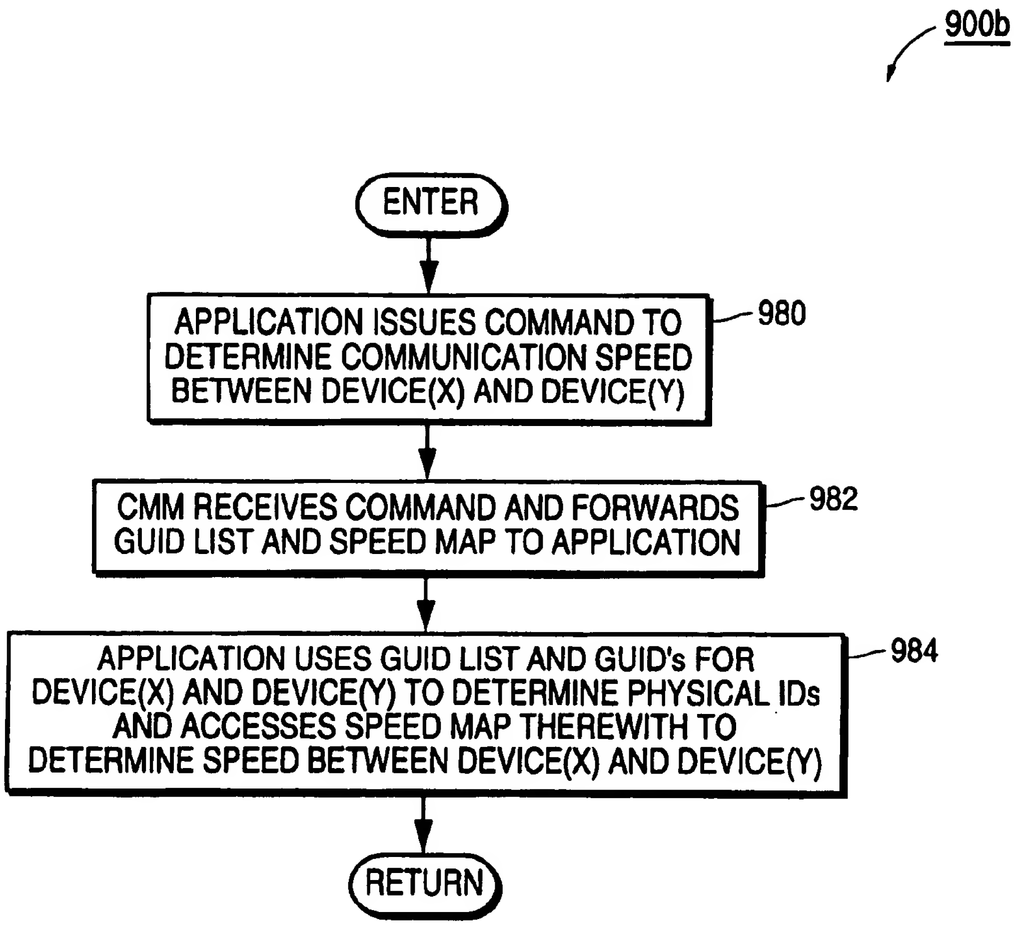

900b
ENTER
APPLICATION ISSUES COMMAND TO DETERMINE COMMUNICATION SPEED BETWEEN DEVICE(X) AND DEVICE(Y)
980
CMM RECEIVES COMMAND AND FORWARDS GUID LIST AND SPEED MAP TO APPLICATION
982
APPLICATION USES GUID LIST AND GUID's FOR DEVICE(X) AND DEVICE(Y) TO DETERMINE PHYSICAL IDs AND ACCESSES SPEED MAP THEREWITH TO DETERMINE SPEED BETWEEN DEVICE(X) AND DEVICE(Y)
984
RETURN

FIG.15D

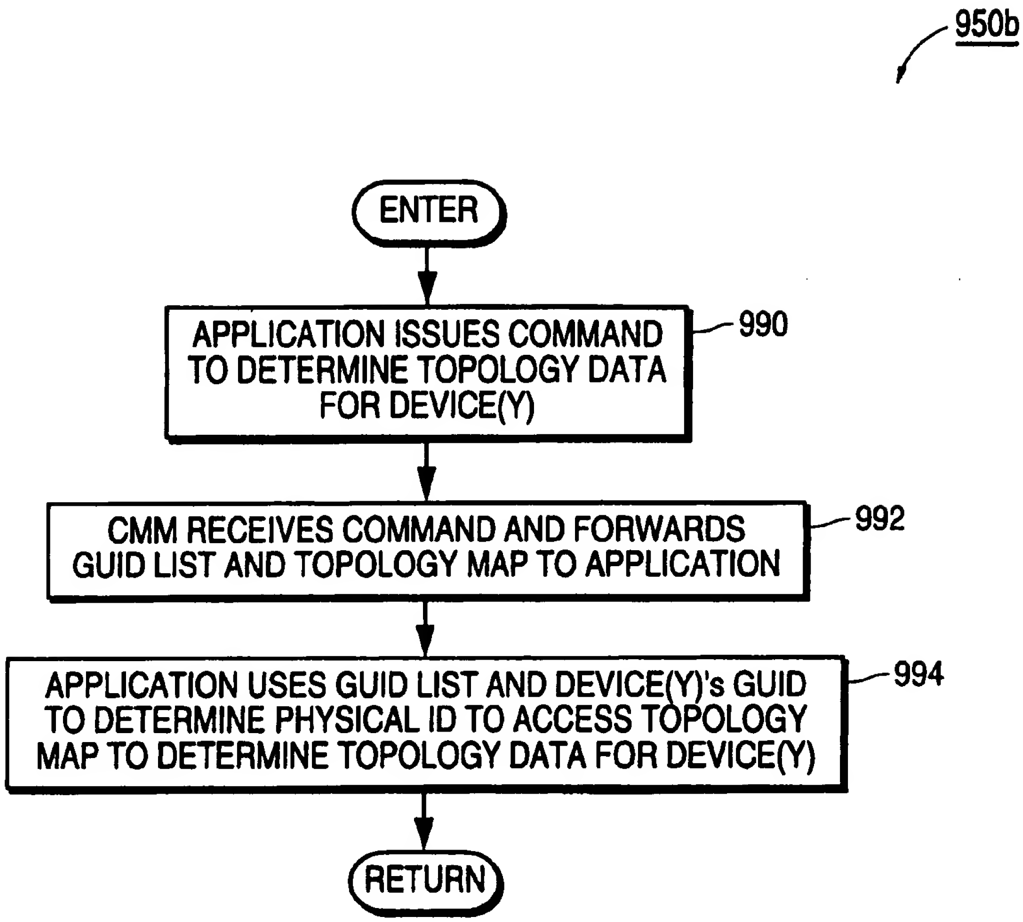

950b
ENTER
APPLICATION ISSUES COMMAND TO DETERMINE TOPOLOGY DATA FOR DEVICE(Y)
990
CMM RECEIVES COMMAND AND FORWARDS GUID LIST AND TOPOLOGY MAP TO APPLICATION
992
APPLICATION USES GUID LIST AND DEVICE(Y)'s GUID TO DETERMINE PHYSICAL ID TO ACCESS TOPOLOGY MAP TO DETERMINE TOPOLOGY DATA FOR DEVICE(Y)
994
RETURN

FIG.15E

METHOD AND SYSTEM FOR PROVIDING A DEVICE IDENTIFICATION MECHANISM WITHIN A CONSUMER AUDIO/VIDEO NETWORK

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to the field of communication systems. More specifically, the present invention relates to equipment for home audio/video electronics.

2. Related Art

A typical home audiovisual equipment complement includes a number of components, e.g., a radio receiver, a CD player, a pair of speakers, a television, a VCR, a tape deck, etc. Components are connected to each other via sets of wires. One component is usually the central component of the home audiovisual system. This is usually the radio receiver, or the tuner/decoder. The control buttons and control switches are usually located on the front of the tuner and in many cases, some or all of these buttons and switches are duplicated on a hand held remote control unit. A user controls the home equipment by manipulating the buttons and switches on the front of the tuner, or alternatively, manipulating buttons on the hand held remote control unit.

As consumer electronic devices have become more capable and more complex, demand for the latest and most capable devices has increased. As new devices emerge and become popular, new devices are purchased by consumers and "plugged" into their home audiovisual systems. The new device is simply plugged into the system alongside the pre-existing, older devices (e.g., cassette tape deck, CD player, and the like). The new device is plugged into an open input on the back of the tuner, or some other device coupled to the tuner. The consumer (e.g., the user) controls the new device via control switches on the new device itself, or via a separate remote control unit for the new device.

As the number of new consumer electronics devices for the home audiovisual system has grown and as the sophistication and capabilities of these devices have increased, a number of problems with the conventional paradigm emerge. One such problem is incompatibility between devices in the home audiovisual system. In addition, where one device is much newer than another device additional incompatibilities may exist. For example, a new device might incorporate hardware (e.g., specific inputs and outputs) which enables more sophisticated remote control functions. This hardware may be unusable with older devices within the system. Another problem is the lack of functional support for differing devices within an audiovisual system. For example, even though a television may support advanced sound formats (e.g., surround sound, stereo, etc.), if an older less capable tuner does not support such functionality, the benefits of the advanced sound formats can be lost. Another problem is the proliferation of controls for the new and differing devices within the home audiovisual system. Each new device coupled to the audiovisual system often leads to another dedicated remote control unit for the user to keep track of and learn to operate.

In view of the above, it is desired to provide a communication architecture in which consumer electronics devices can be integrated. In so doing, the consumer electronics devices can offer and be expanded to include advanced communications and control functionality between themselves not heretofore offered. Within such communication architecture integrating consumer electronic devices ("a consumer's electronics network"), devices can communicate with each other and control one another.

The IEEE 1394 serial communication bus, according to the IEEE 1394 standard, assigns a 6-bit physical identifier value (phy_id) to each device connected to the bus. The IEEE 1394 serial communication bus uses this physical identifier to communicate with a device coupled thereto. Whenever a new device is inserted onto the bus, or an existing device is removed from the bus, or both, a bus reset is caused. The bus reset initiates certain well known bus recovery communications and functions in accordance with the IEEE 1394 standard, including the possible renumbering of the physical identifiers of the devices remaining on the IEEE 1394 bus. That is to say, the physical identifiers assigned to devices on the IEEE 1394 bus are not persistent.

However, it is also desired within the consumer electronics network that high level applications adopt abstract and persistent identifiers for devices coupled to the IEEE 1394 communication bus. Since many data structures (e.g., speed map and topology map) and communications channels are maintained and implemented within the IEEE 1394 standard using physical identifiers, a problem exists for high level applications that use a persistent identifier for each device but need to communicate with devices and/or require speed map and topology map information. What is needed is a mechanism and method operable within a consumer electronic network that provides an IEEE 1394 communication framework, but also provides high level applications with a persistent identifier for devices coupled to the network. What is desired further is an efficient mechanism operable within a consumer electronic network that provides an IEEE 1394 communication framework, but also provides high level applications with a persistent identifier for devices coupled to the network.

SUMMARY OF THE INVENTION

Accordingly, the present invention provides an efficient mechanism and method operable within a consumer electronic network that provides an IEEE 1394 communication framework and also provides high level applications with a persistent identifier for devices coupled to the network. The present invention provides such advantageous functionality utilizing global unique identifiers (GUIDs) assigned to each device of the consumer electronics network. A GUID map is constructed and maintained upon each bus reset that maps GUID values with physical identifier values. These and other advantages of the present invention not specifically mentioned above will become clear within discussions of the present invention presented herein.

A method and system are described herein for providing a device identification mechanism within a consumer electronics based audio/video network. Within the network, several consumer electronics products, e.g., television, VCR, tuner, set-top box (e.g., intelligent receiver/decoder, IRD), DVTRs, PCs, DVD players (digital video disk), etc., can be coupled to communicate together via a standard bus (e.g., IEEE 1394 serial communication bus). This allows devices of the network to control one another and obtain information regarding one another. In one embodiment, the communication architecture used is the home audio/visual initiative (HAVI) format. The HAVI network offers unique advantages consumer electronic vendors because the architecture offers for the home network many of the advantages of existing computer system networks, namely, interconnected devices can share resources and provide open, well defined APIs that allow ease of development for third party developers. HAVI offers extended interoperability.

The present invention provides a mechanism whereby a global unique identifier (GUID) is associated with each

device of the HAVI network. A low level driver, a link driver, constructs a GUID list including a GUID for each device on the HAVI network. The order of the GUID entries in the GUID list (e.g., the index) matches the physical identifiers assigned to the devices by the IEEE 1394 serial bus. Although the physical identifiers can become reassigned on bus reset, the GUID values are constant. Within the present invention, network-based applications use a device's GUID to communicate therewith. Speed map and topology map information is maintained based on the physical identifier information. Therefore translations between GUIDs and physical identifiers are efficiently performed by the present invention and are used for referencing speed map and topology information for an application program or other object.

The 1394 local bus architecture creates a dynamic network within which a 1394 capable device can be inserted (e.g., hot insertion) at any time and be ready for use. Within the local bus system, a device is identified with a 6-bit physical identification number (phy_id) which is assigned by the local bus upon a bus reset. The phy_id for a device can change as new devices are added into or existing devices are removed from the network. To provide higher level applications with a persisting identifier for a given device, the GUID (global unique identifier) is employed by the present invention. The GUID is a unique 64 bit value that contains a vender identification number coupled with a chip series identification number. The GUID is determined (according to IEEE 1212 standards) when an IEEE 1394 capable device is manufactured. Because some bus information, such as speed map and topology map, are referenced by physical identifier values, the present invention provides an efficient mechanism for presenting speed map and topology map information with respect to the corresponding GUID value for a device.

A list of available devices in the network is information that network-based applications generally request periodically. The present invention generates and maintains a GUID list of all devices of the network and orders the GUIDs of the GUID list by their respective physical identifier values. In this manner, the index of a particular GUID within the GUID list is its physical identifier and this index can be obtained and then used to access data from the speed map and topology map data structures which are constructed and maintained with respect to physical identifier values. For instance, if the speed between devicex (GUID1) and devicey (GUID2) is needed, the present invention locates the index values, index1 and index2, for GUID1 and GUID2 using the GUID list. Index1 and index2 are then used to access the speed map data structure and the desired speed data is returned to higher level applications. Alternatively, the entire speed map or topology map can be presented to the higher level application along with the current GUID list and the application can perform the indexing for the desired data.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A illustrates a physical port connection topology of an exemplary device configuration of the home audio/visual initiative (HAVI) consumer electronics network in accordance with the present invention.

FIG. 1B illustrates a local bus connection topology of the exemplary device configuration illustrated in FIG. 1A in accordance with the present invention.

FIG. 1C illustrates a communication channel topology with respect to once device coupled to the HAVI network illustrated in FIG. 1A in accordance with the present invention.

FIG. 2 illustrates internal circuitry of an intelligent receiver/decoder device of the HAVI network of FIG. 1A (acting as a full audio/visual, FAV, node) in accordance with the present invention.

FIG. 3 illustrates a complement of software services made available by a full audio/visual (FAV) node of the HAVI network in accordance with the present invention.

FIG. 4 illustrates communication pathways within the HAVI software architecture in accordance with the present invention.

FIG. 5A illustrates communication flow to the communication media manager (CMM) of the present invention from various software objects.

FIG. 5B illustrates communication flow to various software objects from the communication media manager (CMM) of the present invention.

FIG. 6 illustrates the IEEE 1394 local bus interface (e.g., MPEG/MV Link) with CMM in the HAVI architecture.

FIG. 7A and FIG. 7B illustrates steps performed within the HAVI architecture to implement message communication between devices.

FIG. 8 illustrates communication pathways between an application, a device control module (DCM), the CMM and a link driver in accordance with the present invention.

FIG. 9A is an illustration of a physical identifier indexed topology map data structure used by the present invention.

FIG. 9B and FIG. 9C illustrate, respectively, one dimensional and two dimensional physical identifier indexed speed map data structures as used by the present invention.

FIG. 10A shows a first exemplary device network configuration within the HAVI architecture.

FIG. 10B shows a second exemplary device network configuration within the HAVI architecture.

FIG. 11A illustrates a GUID list as constructed by one embodiment of the present invention for the first exemplary device network configuration of FIG. 10A.

FIG. 11B illustrates a GUID list as constructed by one embodiment of the present invention for the second exemplary device network configuration of FIG. 10B.

FIG. 12A illustrates a general GUID update status list constructed by a second embodiment of the present invention in response to the device network configurations of FIG. 10A and FIG. 10B.

FIG. 12B illustrates an "added" GUID update status list constructed by a second embodiment of the present invention in response to the device network configurations of FIG. 10A and FIG. 10B.

FIG. 12C illustrates a "removed" GUID update status list constructed by a second embodiment of the present invention in response to the device network configurations of FIG. 10A and FIG. 10B.

FIG. 13 is a flow diagram illustrating steps performed by the present invention for constructing a GUID list in response to a local bus reset.

FIG. 14 is a flow diagram illustrating steps performed by the present invention for constructing and communicating the GUID update status lists in response to a local bus reset.

FIG. 15A is a flow diagram illustrating the use of the GUID list in accordance with the present invention for sending a message to a device.

FIG. 15B illustrates the use of the GUID list to provide an index in accordance with the present invention for accessing the speed map to determine communication speed information.

FIG. 15C illustrates the use of the GUID list to provide an index in accordance with the present invention for accessing the topology map to determine topology information.

FIG. 15D illustrates the use of the GUID list to provide an index in accordance with the present invention for accessing the speed map to determine communication speed information where the application references the speed map.

FIG. 15E illustrates the use of the GUID list to provide an index in accordance with the present invention for accessing the topology map to determine topology information where the application references the topology map.

DETAILED DESCRIPTION OF THE INVENTION

In the following detailed description of the present invention, a method and system for providing efficient device identification using a GUID list within a consumer electronics based audio/video network, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be recognized by one skilled in the art that the present invention may be practiced without these specific details or with equivalents thereof. In other instances, well known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

NOTATION AND NOMENCLATURE

Some portions of the detailed descriptions which follow are presented in terms of procedures, steps, logic blocks, processing, and other symbolic representations of operations on data bits within a computer memory (see FIG. 2). These descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, computer executed step, logic block, process, etc., is here, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussions, it is appreciated that throughout the present invention, discussions utilizing terms such as "processing" or "computing" or "translating" or "calculating" or "determining" or "displaying" or "recognizing" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

HAVI ARCHITECTURE GENERALLY

Embodiments of the present invention are operable within a consumer electronics network of the home audio/visual initiative (HAVI) architecture. Aspects of the HAVI architecture network (e.g., "HAVI architecture") are discussed to provide a general framework in which embodiments of the present invention operate.

HAVI is an architecture for inter-operating consumer electronics equipment devices adapted for the home network. The interoperability aspects define an architectural model that allows devices of any vendor to interwork within the home. A feature of the HAVI architecture is the combination of a base set of generic device controls (a device control software module) with a method to extend the base control protocol as new features and devices are deployed.

The HAVI architecture supports a wide range of devices including intelligent receiver/decoders (IRDs), digital video tape records (DVTRs), video cassette recorders (VCRs), personal computers (PCs), digital video disk players (DVDs), etc., communicating via a common messaging system. FIG. 1A illustrates the physical port-to-port connecting configuration **10***a* of an exemplary HAVI network. Several consumer electronics devices ("devices") **12**–**24** are shown connected together with bus segments **30***a*–**30***f* which couple ("plug into") to ports on the respective devices. In one embodiment of the HAVI architecture, the IEEE 1394 serial communication bus standard is used as a local bus platform to provide the common messaging system. The IEEE 1394 serial communication bus carries both commands, status information and well as digital audio and digital video signals between devices.

FIG. 1B illustrates a logical bus configuration **10***b* of the HAVI network of FIG. 1A. As shown in FIG. 1B, all of the devices **12**–**24** of the HAVI network can be viewed as logically coupled to a common IEEE 1394 serial communication bus **30**. Within this bus configuration **10***b*, peer-to-peer device communication is supported. For example, as shown in FIG. 1C, any device (having appropriate capabilities), e.g., device **12**, can send or receive communication packets from any other device in the HAVI network. In the example of FIG. 1C, the set-top-box (e.g., an IRD) can receive messages from or generate messages to any of the other devices **14**–**24** of the exemplary HAVI network.

The interoperability model in the HAVI architecture provides for the following: 1) support for existing devices; 2) a default control model; 2) a means to extend the default control model when new devices or functionality is brought to market; and 4) a common means for device representation (e.g., graphics user interfaces). To achieve the above, the HAVI architecture defines three types of nodes in the home network. A base AV (BAV) node is defined for devices that already exist in the market (e.g., VCR **22** of FIG. 1A). An intermediate AV (IAV) node is defined for simple devices such as camcorders or DVTRs (e.g., camera **14**). A full AV (FAV) node is defined for devices of more resources, such as IRDs or smart televisions (e.g., set-top-box **12**). An FAV node (or device) typically contains enough hardware to host control modules and to support application programs locally. The IAV devices and FAV devices communicate by sending messages over the home network using a generic message passing system described more fully below. When new devices join the home network, they are recognized and added to a global name database (registry). The registry holds information about their characteristics and provides a reference to a handler (e.g., communication point) for that device. Other devices and services are able to query the registry to locate a device and then, using the handler, can interact with the device.

When a device is initially added to the home network, the system queries the device to ascertain its characteristics and

capabilities. Once a device's characteristics are known, the architecture provides two methods of controlling it. The first method, level one interoperability, uses a predefined message set based on IEEE 1394 AV/C-CTS. All IAV and FAV nodes can use this command set to access and control other devices, however, BAV nodes, because they are deployed before the architecture was defined, are controlled using legacy protocols. The level one interoperability provides a default level of control. The FAV nodes act as control nodes and create a local representation of the IAV node, known as a device control module (DCM), that provides an API used to send control commands to the device.

Level two interoperability within the HAVI architecture goes farther and supports future added functionality and new devices. To achieve this, a particular device can carry within its ROM an override DCM that is uploaded from the IAV device, to the FAV device, and replaces the default DCM for the particular device. This override DCM not only contains the basic level one command set for the particular device but also includes vendor specific commands to control advanced features of the device. This model allows the device to inform another object about its particular functionality. Since the override DCM may be loaded onto any vendor's FAV, the format of the DCM is architecture-neutral.

To allow one device to discover the capabilities of another device and to determine which command set to use with that device, a standard device description structure is provided called the self describing data (SDD) structure. The SDD data structure is extensible. It can be a small number of bytes describing the device type, e.g., TV, or VTR, etc. Alternatively, the SDD data structure can be a more complex structure also defining the override DCM and a graphical representation of the device. The graphical representation within the SDD data structure allows an FAV node to present a pictorial representation of the devices in the home network to users.

By defining the graphical representation in a sufficiently generic manner, a device's SDD graphical data can be used in any vendor's product to display (e.g., on a television or monitor within the network) a user interface for that device. This provides an enhanced level of vendor interoperability and also allows a vendor to differentiate a product while maintaining within the general look and feel of the display device. The SDD also enables a control device (the FAV node) to present a general control user interface for all devices in the home network, irrespective of the differences in type and vendor. Self describing data (SDD) is described in copending provisional application serial number 60/054,327, entitled "A Method and Apparatus for Including Self-Describing Information within Devices," filed Jul. 31, 1997 and assigned to the assignee of the present invention. The above referenced provisional patent application is hereby incorporated by reference.

Legacy devices are devices that were built before the HAVI architecture or devices that select not to use the HAVI architecture. The HAVI architecture supports Legacy devices by providing Legacy DCMs to provide protocol conversions for Legacy devices. These Legacy DCMs can contain sufficient knowledge to allow them to support an existing 1 or 2 way control protocol and provide a specific control interface to the devices that conform to the HAVI architecture. A legacy DCM acts as a bridge between the Legacy and HAVI devices. This feature allows the HAVI architecture also to interact with any future device control protocols such as, for example, protocols being used for home energy management or security, etc.

SET-TOP-BOX 12 OF FIG. 2

Any consumer electronics device can be provided with the appropriate hardware to act as an FAV and thereby provide a platform for certain HAVI software (described below). For instance, the set-top-box **12** device of the exemplary HAVI network of FIG. **1A** contains special hardware components that provide an operation platform for software components of the HAVI architecture. Another example of an FAV is a digital television having the required hardware resources to support HAVI software. Certain aspects of the present invention, described below, are discussed in terms of steps executed on a computer system (e.g., processes **400**, **700**, **800**, **850**, **900**a, **950**a, **900**b and **950**b). Although a variety of different computer systems can be used with the present invention, an exemplary general purpose computer system is shown in the set-top-box FAV of FIG. **2**.

Set-top-box **12** of FIG. **2**, in addition to having a video/audio receiver (decoder) unit **106** and MPEG unit **107** also includes an internal address/data bus **100** for communicating information, one or more central processors **101** coupled with the bus **100** for processing information and instructions, a volatile memory **102** (e.g., random access memory RAM) coupled with the bus **100** for storing information and instructions for the central processor **101** and a non-volatile memory **103** (e.g., read only memory ROM) coupled with the bus **100** for storing static information and instructions for the processor **101**. Set-top-box **12** can also optionally include a data storage device **104** ("disk subsystem") such as a magnetic or optical disk and disk drive coupled with the bus **100** for storing information and instructions. Also included in the set-top-box **12** is a bus interface unit **108** for interfacing with the local bus **30** (e.g., an IEEE 1394 serial bus). Set-top-box **12** can operate under a variety of different operating systems (e.g., Windows operating system, DOS operating system, Macintosh O/S), but in one embodiment the Aperios operating system is used.

HAVI SOFTWARE ARCHITECTURE

In one embodiment, the software of the HAVI architecture can be broken into three APIs (AVOS, interoperability and application). The AVOS API is a low level system API that abstracts the common operating system services, e.g., threads, communications, and memory. Interoperability APIs are abstractions of common device control models and provide a basic representation model and a means to extend the control model with vendor specific commands. The application API is a high level API for applications including interactive TV applications, and in one embodiment is based on the DAVIC API set including the Motion Hypermedia Expert Group (MHEG) interactive application engine.

FIG. **3** illustrates the software architecture **200** used in the HAVI architecture in more detail. The architecture **200** contains several generic components to provide abstractions of common facilities in the home network. A communication media manager **250** is provided and this component abstracts from the underlying transport network. In the case of the IEEE 1394, the communication media manager **250** maintains and generates information regarding the speed maps **515** (FIG. **8**) and topology maps **520** which are used by embodiments of the present invention. A DCM Manager **270** is also provided that is responsible for identifying and creating generic DCMs **230** for level one interoperability. The DCM Manager **270** also hosts override DCMs uploaded from devices or other sites. A Graphics Management unit **220** is provided which supplies generic graphics APIs that applications and DCMs can use to present user interface information.

A Stream Manager **335** of FIG. **3** is used to determine the most effective route for AV streams within the home net-

work. A data format manager is included within the Stream Manager **335** and is responsible for providing conversion of AV streams to and from various formats as they flow between devices in the home network. These operations are described below with reference to FIG. **7A** and FIG. **7B**. A Messaging Manager **212** supports a generic messaging mechanism that allows devices to locate and interact with other devices in the home network. A communications media manager (CMM) **250** is a media dependent entity in the HAVI software architecture **200**. The CMM **250** provides abstract services to other HAVI components or application programs in the HAVI network. It also provides media specific transport mechanisms for actual data communication among various devices in the home network. The CMM **250** is responsible for compiling new GUID status updates upon a local bus reset. It is appreciated that each physical communication media contains its own CMM to serve the above purposes. As one example, the CMM **250** for the IEEE 1394 serial bus is described herein further below.

The Event Manager **214**, the Registry **216**, the Stream Manager **335** and the Initialization Manager **210** of FIG. **3** are described separately further below.

FIG. **4** illustrates a data flow diagram **305***a* with the HAVI software architecture **200** including an interface with the local bus HAL layer **330** for local bus **30**. In diagram **305***a*, the top layer is an application program layer ("application layer") **220** which can communicate with a function control module (FCM) layer **314** having AVIC command. The application layer **220** resides within a FAV and can also communicate with device control modules (DCMs) **230** and with the Registry **216**. The application layer **220** communicates with a Stream Manager **335**. The DCMs **230** are controlled by the DCM manager **270** which can communicate with the Event Manager **214**. The Event Manager **214** communicates with the CMM **250**. The Stream Manager **335** allows communication between the application layer and the CMM **250**. The AV Messenger unit **212** also communicates with the CMM **250**. The CMM **250** provides an interface with the local bus interface **330** via a link driver unit **320**. The link driver unit **320** is responsible for compiling new GUID lists upon a local bus reset. Higher level applications are located with block **310** and lower level applications within block **312**. A subset of this communication diagram **305***a* is illustrated in FIG. **8**.

FIG. **5A** is a diagram **340***a* that illustrates communication channels that are available for communicating to the CMM **250**. The application layer **220** communicates with the CMM **250** to obtain bus generation information, to obtain the GUID list, to obtain the speed map **515** and to obtain the topology map **520**. The following APIs are used to perform these functions: getGUIDList( ); getBusGeneration( ); getSpeedMap( ); and getTopologyMap( ). The ilink transportation adaptation module, TAM, **212***b* communicates with the CMM **250** for asynchronous requests and asynchronous responses. The digital I/F controller **335***a* communicates with the CMM **250** for asynchronous requests, asynchronous responses, channel allocations, channel deallocations, bandwidth allocations and bandwidth deallocations. The Stream Manager **335** utilizes GUID information to establish point to point communications via the digital I/F controller **335***a*. The digital I/F controller **335***a* communicates with the CMM **250** using the following APIs: allocateChannel( ); deallocateChannel( ); allocateBandwidth( ); and deallocateBandwidth( ). The isochronous data transceiver **230** is a DCM and communicates with the CMM **250** using the following APIs: isoOpen( ); isoClose( ); isoAttach( ); isoDetach( ); and isoControl( ).

FIG. **5B** is a diagram **340***b* that illustrates communication channels that are available for communicating from the CMM **250** to other objects. The CMM **250** communicates bus reset indications to the Event Manager **214**. The CMM **250** also communicates status information regarding the set of current devices on the local bus **30** including information as to which devices are added and which devices have been removed from the local bus **30** since the last GUID list was constructed. The CMM **250** also communicates bus reset information to the ilink TAM **212***b* as well as GUID information, asynchronous receive, and isochronous receive. This same information is also communicated from the CMM **250** to the isochronous data transceiver module **230**. Information regarding which devices are coupled to the local bus **30** is also communicated from the CMM **250** to the digital I/F controller **335***a*.

FIG. **6** illustrates an exemplary interface **360** between the CMM **250** and the local bus **30** for IEEE 1394 interfacing. A 1394 Bus Manager **370** is also shown. The CMM **250** contains a copy of the speed map **515** and the topology map **520** (described further below) and and IEEE 1394 Bus Manager **370** contains control/status registers (CSRs). The isochronous resource manager **372** contains information regarding the bus master identification and the available channels and bandwidth for communication via isochronous registers (CSRs). The node controller **374** contains a configuration ROM as well as node control registers (CSRs). Units **250**, **372** and **374** constitute the serial bus management unit **370**. Unit **370** communicates with 1394 HAL interface (I/F) layer **330**. The transaction block **378** processes read/write/lock transactions, tracks pending transactions and controls retry protocol operations with a busy/timeout register. Data transmission takes place within the link layer **380** and the I/F (CFR) unit **385**.

DCM 230 AND DCM MANAGER 270

A description of the DCM **230** and the DCM manager **270** of FIG. **3**, as well as other aspects of the HAVI software architecture **200** are described in copending U.S. patent application Ser. No. 09/003,119, entitled "A Home Audio/Video Network with Two Level Device Control", by R. Lea and A. Ludtke, filed concurrently herewith, attorney docket No. SONY-50L2278, assigned to the assignee of the present invention and hereby incorporated herein by reference.

THE COMMUNICATIONS MEDIA MANAGER 250

The CMM **250** (e.g., for the IEEE 1394 bus) of FIG. **4** provides two levels of APIs. One is the high-level API that interacts with other HAVI components to offer abstract services such as topology map **520** and speed map **515** as well as generating HAVI related events. The other level of APIs is the low-level API that provides a basic transport mechanism for both asynchronous and isochronous data transfers. Normally, high-level APIs are available to any HAVI entity or applications residing locally on this same node, while the low-level APIs presents the primary interface to the IEEE 1394 bus **30** so that specific protocols can be built on top of the CMM **250**. To access high-level APIs, the HAVI message system (e.g., AvMessenger **212**) is used, whereas access to low-level APIs can be accomplished through direct calls to the CMM **250**.

The messaging used to access high-level APIs through HAVI message system is done using a format having a multi-byte selector followed by function dependent parameters. The function selector determines which service to use

from the CMM **250**, and the parameters section provides all required parameters for the specified function. To invoke low-level APIs, only the parameters section is necessary.

The IEEE local bus **30** is a dynamically configurable network. After each bus reset, a device can have a complete different physical identifier (phy_id) than it had before the bus reset. If a HAVI component or an application has been communicating with a device in the HAVI network it generally desires to continue the communication after a bus reset even though the device may have a different physical identifier. To identify a device uniquely regardless of frequent bus resets, the present invention provides the Global Unique ID (GUID) which is used by the CMM **250** and other HAVI entities. A GUID is a multi-bit number (e.g., 64-bits in one implementation) that is composed of 24 bits of node-vendor identification and 40 bits of chip (series) identification. While a device's physical identifier may change constantly, its GUID is permanent within the HAVI architecture. The CMM **250** of the present invention makes device GUID information available for its clients using a GUID list discussed further below.

The CMM **250** abstracts the underlying device interconnection mechanism, providing a common set of API's to describe the capabilities of the bus architecture. In one embodiment, the CMM concept is driven by the IEEE 1394 bus model, but is sufficiently abstract to provide a general service for interconnection management. The CMM **250** provides APIs for 4 four basic concepts. (1) Channel: a channel is a logical data connection that a bus can support. For technology such as IEEE 1394, this maps directly to hardware support for multiple channels. For technology that only supports a single data connection, then this degenerates to a single channel. (2) Bridge: a bridge is a connection between two bus technologies and allows channels to span different bus technologies. (3) Node: a node is a physical endpoint for a channel and corresponds to a unit or subunit in the HAV architecture. (4) Bandwidth: this is a metric of how much data can be carried on a bus at any one time. Bandwidth can be assigned to channels to support the required data rates for whatever data the channel is delivering.

For those bus protocols which do not support these concepts, the results of API calls to their CMMs have limited results. For example, if a particular bus technology does not support multiple simultaneous data transfer actions, then it most likely reports only a single "channel," even though it does not really support channels. The CMM **250** maps transactions to the channel model.

There is one CMM **250** for each physical interface (e.g. 1394, Control A1, TCP/IP) on a FAV node. Because there can be more than one kind of CMM **250**, and because there can be many of them in existence, each CMM **250** has a module_type attribute attached to its entry in the Service Registry **216** database. The definition of this attribute can be as follows:

module_type=busManager

module_data={1394, Control A1, TCP/IP, etc.} These bus types will have numeric identifiers defined for them.

Each CMM **250** contains the general set of API's for bus management, as well as protocol-specific API's. If a particular bus technology supports features that are above and beyond the general model, then it can subclass the CMM base class and add the new functionality. For example, the 1394 CMM **250** allows access to the 1394-specific bus attributes such as the isochronous resource manager. Those clients who are 1394-aware, and need such capabilities, can go through the 1394 CMM **250**.

One of the advanced features the 1394 bus provides to the HAVI architecture is its support for dynamic device actions such as hot plugging (device insertion or power up) and unplugging (device removal or power down). To fully support this to the user level, high level software clients need to be aware of these environment changes and the present invention provides this information. The CMM **250** works with the Event Manager **214** to detect and announce these dynamic bus changes using device status tables (FIG. **12A**, FIG. **12B**, FIG. **12C**). Since any topology change within 1394 bus will cause a bus reset to occur, the CMM **250** is informed of these changes and notifies the Event Manager **214** of these changes along with the information about devices that have disappeared as well as those that have become available. The Event Manager **214** then distributes the related event to all interested HAVI entities or applications that previously established the proper call back handlers.

There are two basic types of events that are generated from the CMM **250** to support dynamic updating of available devices in the network. One is NEW_DEVICE which indicates a new device, and the other is GONE_DEVICE, indicating a device has been removed. To increase efficiency, the CMM **250** generates device status tables with this information. Whenever a device is added or removed from the network, a bus reset is generated and CMM **250** finds and passes a status table of new/gone devices to the Event Manager **214** if the Event Manager **214** has requested such service. Since NEW_DEVICE or GONE_DEVICE also indicates the occurrence of a bus reset, the Event Manager **214** may also create a BUS_RESET event based on the above information. It should be noted that all these events are generally "local," which means that the Event Manager **214** does not generally send/broadcast such events to any other nodes.

The CMM **250** also provides topology, speed maps, and other environment descriptions to its clients. The topology map **520** (FIG. 9A) depicts the connectivity among physical devices. Within the present invention, it can be used to build a human interface that helps the user understand how devices are connected and how certain features may be used. The speed map **515** (FIG. 9B, FIG. 9C) provides information on the possible maximum speed that can be used for data transmission between any two devices in the network. Speed maps provide information about the connectivity and performance of sections of the HAV network. It can be used to analyze the current connection scheme and give the user helpful suggestions for improving the performance of devices by rearranging the connections. If attributes such as the topology map **520** cannot be automatically generated, then a dialog with the user may be required. When a topology request is made to a CMM for a protocol that does not support automatic topology detection, the request can trigger such a dialog. After interacting with the user to determine how its devices are connected, the CMM returns the requested information.

Topology map **520** and speed map **515** may change constantly because their organization (e.g., data order) is based on the physical identifiers of the devices on the network and these values can become reassigned due to the allowed dynamic insertion or removal of devices. To identify the right "version" for these maps, a bus generation number may be used. Newer bus generation values signify the change of bus configuration. It may also be used to check if a pending operation is "stale". Since transactions between

devices require the existence of those devices, it is possible that a pending transaction has not yet completed and one or more devices involved have gone away. When this happens, a new bus generation number will be generated and the related process can check against this new generation number to make proper clean up for the stale transaction.

Bus generation number, topology map 520, and speed map 515 remain unchanged between bus resets. To improve efficiency, the CMM 250 may optionally cache them internally and keep them updated only when bus reset occurs.

Another feature of the IEEE 1394 bus 30 is its capability to send timingcritical stream through its isochronous channels. To send data in isochronous mode, channel number and proper bandwidth need to be allocated from the bus. The CMM 250 provides such services to allocate/deallocate channel numbers and bandwidth for its clients. It is also responsible to restore these resources for its clients right after each bus reset so that any existing isochronous data transaction will not be disrupted because of the unavailability of these resources.

The CMM 250 also provides a transport mechanism to actually transfer the data, which is part of low-level services. There are two types of data transactions with IEEE 1394 bus 30. One is asynchronous, the other isochronous. Typical asynchronous transfer includes quadlet/block write, quadlet/block read, and lock operations. Typical isochronous data transfer normally involves allocating channel/bandwidth, opening isochronous data port, attaching data buffer(s), and activating data transfer. These services provide a fundamental transport mechanism for a 1394 based HAVI system being functional.

The following APIs are provided by IEEE 1394 specific CMM 250 within the present invention:

1) CMM::enrollComeNGo( )

Status: enrollComeNGo(ObjectID oid, (void *) (*) callback_handler)

This command is used by the Event Manager 214 to get a list of new and removed devices when bus reset occurs. The Event Manager 214 enrolls such request to the CMM 250 to get the service. The parameter oid is the caller's object identifier and callback_handler is the callback function provided by the caller. The caller (typically Event Manager 214) enrolls its OID and a callback handler to the CMM 250 so that when bus reset occurs, the CMM 250 invokes the callback function with a list of devices (device status table) that are either new or gone.

2) CMM::getGUIDList( )

Status: CMM 250::getGuIDList(GUID *guid)

This command gets the GUID for all active devices in the network. The parameter guid is a pointer to a the GUID list.

3) CMM::getBusGeneration( )

Status: CMM::getBusGeneration(uint32 *bus_en_number)

This command gets the current local bus generation number from the network.

The parameter bus_en_number is the current bus generation number.

4) CMM::getSpeedMap( )

Status: CMM::getspeedMap(uint32 bus_en_number u_char *spd_map)

This command gets the speed map 515 which is associated with the bus generation number specified. The parameter bus_en_number is the current bus generation number, spd_map is a pointer to the speed map 515

5) CMM::getTopologyMap( )

Status: CMM::getTopologyMap(uint32 bus_en_number, uint32 * top_map)

This command gets the topology map 520 that is associated with the bus generation number specified. The parameter bus_en_number is the current bus generation number and top_map is a pointer to an array of integer numbers, each representing a selLID packet from a 1394 node.

6) CMM::allocatechannel( )

Status: CMM:: allocateChannel(uint32 chan_no)

This command allocates a channel specified by chan_no. If chan_no is all ones (Ox FFFFffff), any channel available will be allocated and returned. The parameter chan_no is a channel number to be allocated.

7) CMM::deallocatechannel( )

Status: CMM::deallocateChannel(uint32 chan_no)

This command deallocates a channel specified by chan_no. The parameter chan_no is the channel number to be deallocated.

8) CMM::allocateBandwidth( )

Status: CMM:: allocateBandwidth(uint32 bandwidth)

This command allocates the bandwidth specified by bandwidth. The parameter bandwidth is the bandwidth to be allocated.

9) CMM::deallocateBandwidth( )

Status: CMM::deallocateBandwidth(uint32 bandwidth)

This command deallocates the bandwidth specified by bandwidth. The parameter bandwidth is the bandwidth to be deallocated.

10) CMM::asyncWrite( )

Status: CMM::asyncWrite(GUID guid, u_int64 remote_offset, u_char *data, uint32 data size, uint32 pk_format)

This command sends an asynchronous write request to the remote device, which is identified by guid. The parameter guid is the target device's GUID, remote_offset is the target device's memory offset, data is a pointer to data to be sent, data_size is the size of data (in byte) to be sent and pk_format is the format to use, either quadlet or block for data transfer.

11) CMM::asyncRead( )

Status: CMM::asyncRead(GUID guid, u_int64 remote_offset, u_char *data, uint32 data_size, uint32pk_format)

This command sends a asynchronous read request to the remote device, and returns the data read back. The parameter guid is the target device's GUID, remote_offset is the target device's memory offset, data is the pointer to data to be received, data_size is the size of data (in byte) to readback from remote site, and pk_format is the format to use, either quadlet or block for data transfer.

12) CMM::asyncLock( )

Status: CMM::asynclock(GUID guid, u_int64 remote_offset, u_char *old_data u_char *new data, uint32 data_size uint32 exLcode)

This command makes a lock operation on the specified address in the remote device. The parameter old_data is used for compare and swap operation. The parameter guid is the target device's GUID, remote_offset is the target device's memory offset, old_data is the pointer to old data, new_data is the pointer to the new data, data_size is the size of data (in byte) to be sent, and ext_code is the extended code to be used for the lock operation

13) CMM::isoOpen( )

Status: CMM::isoOpen(uint32 direction, uint32 bus_speed, uint32 bandwidth, uint32 *port-no)

This command opens an isochronous port for either sending or receiving operation. The actual port opened is returned in port_no. The parameter direction is the sending or receiving operation, bus_speed is the requested bus speed, bandwidth is the maximum bandwidth to be used, and porLno is the port number successfully opened.

14) CMM::isoClose( )

Status: CMM: :isoClose(uint32 port_no)

This command closes the port specificed. The parameter port_no is the port number to be closed Close the port specified.

15) CMM::isoAttach( )

Status: CMM::iosAttach(uint32 porLno, u_char *buffer)

This command attaches a user buffer to the specified port. The parameter porLno is the port number to be used and buffer is the data buffer.

16) CMM::isoDetach( )

Status: CMM::iosDetach(uint32 port_no, u_char *buffer)

This command detaches a user buffer from the specificed port. The parameter port_no is the port number to be used and buffer is the data buffer.

17) CMM::isoControl( )

Status: CMM::isoControl(uint32 chan, uint32 trigger)

This command controls the actual data transfer. It starts immediately or on some synchronization bit. The parameter chan is the channel to be used for the transfer and trigger is the method to use to start the traffic.

In addition to the basic four abstractions discussed above (Node, Channel, Bridge and Bandwidth), the following concepts describe the bus model. The topology map **520** is a connected graph which mirrors the physical connections between nodes on the network. For IEEE 1394, the map is generated automatically. For other protocols, manual assistance from the user may be required. The overall system topology is an amalgamation of individual bus topologies. The generation number provides a way to measure whether a pending operation is "stale." Since transactions between devices depend on being able to access those devices, it is possible that a pending transaction has not yet completed and one or more of the devices involved have gone away. When this happens, any modules that care about the bus generation react accordingly.

The generation value is based on the ability to determine that the bus configuration has changed. For protocols such as IEEE 1394, the generation is critical because of hot plugging support. For others, if hot plugging support can be simulated, then the generation number can be managed as well. If no hot plugging support can be created, then the generation number will always default to "1," meaning that the bus has not changed since the host system was initialized. A bus can allow a certain maximum number of nodes (devices) to be connected and individually addressable at any given time. At any given time, the CMM **250** is informed of how many devices are currently attached and addressable (number of nodes)

As described more fully below, the CMM **250** provides bus reset or change notification for buses that support dynamic connection. After a bus reset is detected, this module assigns new "opaque" ID values to all devices that have just appeared, determines what devices have gone away, invokes the DCM Manager **270** module to create new DCMs, and posts a bus_change notification event to the Event Manager **214**, which notifies all registered clients about the reset. In accordance with the present invention, it also provides enough information for the clients to determine what devices have disappeared and about any newly discovered devices. The CMM **250** works with the Event Manager **214** to detect and announce dynamic bus changes that do not trigger system-wide interrupts or events, thus bringing some of the advantages of 1394 technology to other bus protocols.

The CMM **250** provides clients with the physical topology map **520**, in a manner that does not require clients to have bus-specific knowledge. The topology map **520** can be generated automatically for those control protocols which support it, or can be generated with user interaction and stored for later use for those protocols that do not support automatic generation. Because topologies can be quite diverse, the CMM **250** provides a topology iterator function similar to that of the Service Registry **216**. This function allows clients to visit each node in a connection-specific manner, with a simple API to move forward or backward through the net. The client can determine simply that this device is connected to that one, which is connected to that one, etc. For clients who simply need to know about all devices in a topology, with no concern for connection patterns, the GUID list is provided. Other API's allow the query to determine if a given moduleID (the identification of a DCM which represents a physical device) is represented in the topology, etc.

The CMM **250** also provides logical connection information. This is a connection model that specifies data flow sources and sinks. The API allows inquiries such as "given DCM module ID X, tell caller all of the DCMs that are listening to it." Other variations on this inquiry model include the ability to ask for all currently active streams of a particular data type (e.g. MPEG, SD), etc.

EVENT MANAGER 214

The Event Manager **214** of FIG. **4** is designed to support a one-to-many model of communication. It is built above the basic messaging system **212**. Objects register interest in events which are then delivered to their event call back entry. For instance, objects interested in device status information register special call back handlers for this information with the Event Manager 214. Objects can attach event filters to the callback entries that allow them to filter out events that they are not currently interested in. For example an object may register for all events from display devices, but filter out those display devices it is not currently using.

Events are broadcasted by the messaging system **212** and delivered to all objects that have registered an interest in the event. To achieve this, objects register their interest with their local Event Manager **214**. The messaging system delivers events to all known Event Managers **214** which are then responsible for ensuring that registered objects are informed about events. Filters can be attached to Event Managers **214** to allow objects to filter out the events as they arrive. This gives an object the flexibility to register interest in an event, but have control over how each event is used. For example a filter could be used to specify that event X is only of interest if it occurs after event Y. The Event Manager **214** tracks filters and provides an API that allows applications and DCMs to build sophisticated event filtering chains.

STREAM MANAGER 335

The Stream Manager **335** of FIG. **4** provides an API for configuring end-to-end isochronous ("streaming") connections. Connections may be point-to-point or utilize unbound sources or sinks. The responsibilities of the Stream Manager **335** include: configuration of connections; requesting allocation of network resources; maintenance of global connection information; and support of high-level stream operations such as plug type checking, splicing and bridging.

A Stream Manager **335** runs on each FAV node. The interface is based on the IEEE 1212 unit directory model and the IEC 6 1883-FDIS plug model (HAVI refers to units and subunits as devices and functional components respectively). Point-to-point connections and broadcast con-

nections are supported. In IEEE 1394, a broadcast connection is one in which either the source or sink is not specified when the connection is requested. Streams do not cross transport boundaries, but bridge devices can be used to feed one stream into another. Streams originate and terminate at functional components (rather than devices). For IEEE 1394 networks, the Stream Manager **335** is responsible for allocating PCRs (plug control registers) and requesting bus bandwidth. A standard naming scheme is used for stream types, transport types and transmission formats. The stream type and transport type naming schemes are specified by HAVI, transmission formats are the FMT/FDF values used in IEC 1883 Common Isochronous Packets.

A stream type identifies a media representation, this can be a data format (for digital media) or signal format (for analog media), e.g. CD audio, NTSC. A transport type identifies a transport system, two examples are IEC 1883 (for connections running over 1394) and "internal" (for connections internal to a device). A transmission format identifies the data transmission protocol used to send a stream type over a transport type. For IEC 1883, the transmission format corresponds to the FMT and FDF fields in the Common Isochronous Packet (CIP) header.

A stream as used by the Stream Manager **335** is based on the isochronous data flow model from IEC 1883 with three differences. The first difference is that streams do not restrict the number of sources and so can have multiple sources provided the transport supports this form of connectivity, for example, a UDP "event" stream being fed by many processes. The second difference is that an IEEE 1394 data flow starts at a device output plug and ends at a device input plug, while a stream typically starts at a functional component output plug, goes to a device output plug, then a device input plug, and ends at a functional component input plug. The final difference is that streams are typed, e.g., for each stream there is an associated stream type identifying the data

The following summarizes the properties of streams: 1) a stream is associated with a globally unique stream identifier; 2) a stream is carried over a single channel; 3) output functional component plugs ("plugs") and input plugs can join a stream (this can also be stated as "connect to the channel"); 4) a channel can be connected to zero or more output plugs and zero or more input plugs; 5) an input plug can join at most one stream; 6) an output plug can join many streams (for example a functional component output plug attached to several device output plugs); 7) a stream can be created with no output plug, but no data will flow over the stream until an output plug has joined; and 8) the upper limit on the number of plugs that can be connected to a channel is transport type dependent.

A stream can be viewed as a set of plugs. The following rules apply to connections within a device. Functional component input plugs can have only one connection (from a functional component output plug or a device input plug). Functional component output plugs can have many connections (to functional component input plugs and/or device output plugs). Device output plugs can have only one connection from a functional component output plug. Device input plugs can have many connections to functional component input plugs.

There are three procedures for adding or removing plugs from this set: 1) point-to-point procedure (Connect, Disconnect); 2) broadcast-out procedure (StartBroadcast, StopBroadcast); and 3) broadcast-in procedure (Tap, Untap). Connect creates a point-to-point connection between two plugs (e.g., a connection from an output plug to the channel and a connection from the channel to the input plug); it adds an output plug and an input plug to the stream (assuming neither are already members). StartBroadcast creates a "broadcastout" connection (e.g., a connection from an output plug to the channel); it adds an output plug to the stream (assuming it is not already a member). Tap creates a "broadcast-in" connection (e.g., a connection from the channel to an input plug); it adds an input plug to the stream (assuming it is not already a member). Disconnect, StopBroadcast and Untap drop the connections created by Connect, StartBroadcast, and Tap respectively. They remove plugs (if not used by other connections within the stream).

Streams allow the overlaying of connections as described in IEC-1883, e.g., a plug may have many connections (both point-to-point and broadcast) to the channel used by the stream. Following the connection semantics of IEC-1883, the Stream Manager **335** decides when to remove a plug from a stream by examining the plug's point-to-point connection counter and a broadcast flag (which indicates whether or not the plug participates in a broadcast connection). A plug is removed from a stream when the point-to-point connection is 0 and the broadcast flag is false. The underlying network resources used for a stream are allocated when the first plug is added to the stream and released when the last plug is removed. In the case of a channel connected to several output plugs, the manner in which the resulting data flow is delivered is transport type dependent and the Stream Manager **335** does not guarantee that all input plugs will receive data in the same order.

The Stream Manager **335** maintains a map of all stream connections within the home network. This map includes internal connections (those within devices) and external connections (those over transport systems). The Stream Manager **335** builds the Global Connection Map upon initialization, after bus reset, and periodically while running (in case new connections have been added or removed by non-HAVI applications).

An AV network can take on many topological configurations, depending on how the user has made the connections between devices. For example, an IEEE 1394 bus allows any topology except a loop. The Stream manager **335** of FIG. **3** works with the CMM **250** to provide services to assist with routing data from source to destination. This can include many nodes in between. In the event that the source and destination involve different data types, or are separated by a "data type barrier," it will work with a data format manager and Service Registry **216** to handle automatic or requested data translation services as well. Some routing can be performed automatically, based on the capabilities provided by the underlying bus architecture. For example, the topology of the IEEE 1394 bus can be precisely determined, and therefore automatic routing between directly-connected 1394 devices can be achieved. However, other connection mechanisms can require interaction with the user, either to assist the user with making well-known connections or to have the user indicate to the controlling software how devices are connected.

One of the most significant attributes of the IEEE 1394 technology is isochronous data flow. Additional services provided by the Stream manager **335** for this kind of data include buffer allocation and management. Buffer management includes the provision of a consistent notification mechanism to let the client know when data is available, so that it can be processed. While isochronous data is flowing into a client, various memory buffers will be filled with that data. As each buffer is filled, the client may want to know so that it can handle the data acquisition process from that point forward.

In addition, buffer management can be simplified for clients by having the appropriate service modules partition memory in a manner that is optimized for the data being captured. For example, the client can allocate one large space of memory, then specify that it will be used for capturing a stream of video data. It is possible that the optimum configuration of this memory is to separate it into scan line- or frame-sized segments, so the client can receive one complete line or frame of video on each notification. Raw audio and MIDI data will likely have different optimum segment sizes. Such services require close cooperation between the Route and Data Format Managers, and various service modules.

Refer to FIG. 7A and FIG. 7B where an example process **400** is shown for setting up a data flow. A client wants to establish a connection between two devices, so at step **410** it calls to establish an external connection of one of the two DCMs that represent those devices. It passes the moduleID of the other device's DCM as a parameter. At step **415**, the DCM that was called then calls the Stream Manager (SM) **335** to assist with making the connection. It passes both the source and destination DCM module IDs as parameters. At step **420**, the SM **335** analyzes the source and destination IDs, finds that they are in different nodes (this step may be carried out by some other entity before invoking the DFM). At step **425**, the SM **335** asks the CMM **250** of the source node for the topology map **520** for its network. At step **430**, the SM **335** analyzes the topology map **520** to find the destination node.

At step **435** of FIG. 7A, if the destination node is on the topology map **520**, then no transport protocol translation is necessary and step **445** is entered directly. If the destination node is not on the map, then at step **475** (FIG. 7B) the SM **335** asks the Registry **216** for the module with the destinationID, to look at the "transmission_protocol" attribute, or alternatively, it gets the bus manager ID for that module/node. At step **480**, the SM **335** then finds a transmission protocol service module, and sets up the conversion process (see separate example for these details). At step **485**, the above process is repeated if multiple transports need to be bridged.

At step **445** of FIG. 7A, the SM **335** analyzes the connection paths to find the best route. If no deterministically optimal path can be determined, then a best guess is accepted. This is done for the entire path from source to destination, including intermediate or cross-network boundaries. All necessary connections are made (in the case of dynamic buses such as 1394). At step **450**, the SM **335** analyzes the input data formats for source and destination. At step **455**, if the formats are the same, then no format conversion is necessary. At this point (**465**) the data flow route **400** is complete.

If conversion is necessary, then at step **460**, the SM **335** asks the Registry **216** for the appropriate format converter based on input and output format and sets up the conversion process. Note that several converters can be chained together to achieve the final result from original source to desired destination formats. The data flow route **400** is then complete.

SERVICE REGISTRY 216

The Service Registry **216** of FIG. **4** provides a mechanism to locate devices and services that are available in the HAV network. Since all devices and services are represented by objects, the Service Registry **216** allows any object to advertise itself. Typically, resident on an FAV, the Service Registry **16** contains information on local service objects, remote service objects and local and remote devices. The Service Registry **216** operates by providing an API that allows objects to register their unique identifier, a name, and a set of attributes that define their characteristics including connection point information. Any object wishing to locate a particular service or device object can do so by querying the Service Registry **216** for the appropriate DCM.

Attributes associated with an object may be static or dynamic, and can be subsequently changed if desired. Attribute querying is carried out by specifying a set of required attributes which then returns a list of objects that match the required attributes. Identifiers returned as a result of a query are not guaranteed to be valid. This can happen in one of two ways. First, if a device has posted its availability into the registry and then has gone off-line. If the query arrives between the time of going off-line and actually withdrawing the service from the registry, then the identifier will be stale. Second, if a query returns a set of identifiers, the results may be held within an object for an extended period of time. When they are finally used, it may be that the service has been withdrawn from the registry.

The Service Registry **216** is a distributed service. Any object that registers with its local Service Registry will be accessible via any other instances of the Service Registry **216** in the HAV network. Each Service Registry **216** works with others to ensure that entries are replicated or available when required. However, for both performance and resource reasons, it is often the case that an item registered in one registry will not be visible in another, although the system will try to ensure this. This implies that a simple query to a local Service Registry **216** may return the local registry's view of the global state. It will always be possible to cause a remote query to be carried out on behalf of the query to ensure that the global state of the system is queried explicitly.

Queries to the Service Registry **216** return object identifiers usable as end point for message communication. These identifiers may refer to DCMs, services, or any other entity in the system accessible via the messaging system. The format of the queries allows both sophisticated queries that iterate over the global registry or simple queries that are confined to a local registry.

MESSAGING 212

The messaging system **212** of FIG. **3** is designed to provide a reliable high level messaging service that supports the transport of 4 different types of messages. (1) Commands: these flow between objects in the system and are used to access functionality. When the target object is a device then these messages are sent by a DCM and contain actual device control messages. (2) Events: events are used as a general purpose asynchronous communication mechanism that allows both devices and service modules to communicate information. Although the messaging system **212** is used to carry these events they are under the control of the Event Manager **214** which uses a broadcast feature of the communications infrastructure to send events to those who are interested. (3) Errors: errors are similar to events except errors are delivered to the source of the command that caused the error and then broadcast. (4) Status: these messages are designed to be used in response to a query.

The message system **212** provides both a synchronous and an asynchronous send allowing services to operate in either mode. The messaging system **212** defines a minimum capability to allow interoperability at the message level. This

includes the format of identifiers, the format of message identifiers and the format of messages.

The following illustrates an exemplary implementation. Local identifiers are 56 bit and are guaranteed to be unique to a particular host. Network identifiers are 128 bits and consists of a 64 bit node identifier, an 8 bit node type and the 56 bit local identifier. The generic format for remote messages is a 4 octet type field followed by a parameter block. This is referred to as the message payload. The mechanism to support local messaging is implementation dependent and what is required is that local identifiers guarantee consistent delivery of the message payload. Remote messaging takes a message payload and encapsulates it in a network message. The network message format is header plus message payload. The network message header is 96 bits and consists of a reserved 6 bit field, a 2 bit packet type, a 56 bit local identifier and a 32 bit message identifier. Depending on the actual transport mechanism, this remote message packet will be encapsulated in a transport specific message, e.g. IP or IEEE 1394.

INITIALIZATION MANAGER 210

The Initialization Manager **210** of FIG. **3** is used to bring an IAV or FAV node up to initial status. In the case of an IAV node this manager **210** will initialize the messaging **212**, event **214** and registry **216** service. For the FAV node it will also invoke initialization routines for the DCM manager **270** and any pre-loaded services.

APPLICATION INTERFACE 220

The Application Interface **220** of FIG. **4** is a proxy to the messaging system **212** and provides a means for applications to access services and devices in the AV architecture. The Application Interface **220** is not necessarily a component of the AV architecture, but rather a means of allowing an arbitrary application to work with the AV architecture. The mechanism used to provide the application object depends on the location of the application and the execution environment it uses. Applications can be downloaded from external services provided and made resident (e.g., instantiated) within an intelligent device (FAV) of a HAVI network. An Application so downloaded can query the Service Registry **216** to determine connection points for controlling devices of the HAVI network.

There are 3 general cases. The first case is local application running native on the FAV node. These types of applications are designed to make use of the native OS and hardware features of the FAV node. They are created externally to the AV architecture and use the application interface to gain access to the AV architecture services. The exact means to achieve this are system dependent, but a library linked into the application is a typical approach.

The second case is local application running above the FAV interoperability language run-time. In this case, the AV architecture uses an architecture neutral run time to support level 2 interoperability. This run time can also be used to support arbitrary applications written to be hosted on the run time. Such applications are managed by the AV architecture and will typically access the AV services via a run time supplied calling mechanisms.

The third case is remote application. These applications may be running on other devices in the home network such as home personal computers (PCs). These applications need access to the AV architecture and do so by using the basic message passing model of the system. This implies that these applications need a partial implementation of the messaging infrastructure resident on their host node.

GRAPHICS MANAGER 218

The Graphics Manager **218** of FIG. **3** provides a high level API for the management of the user interface (UI) associated with devices. The role of the Graphics Manager **218** is to support the User Interface model by providing a graphics API that is semantically rich. Low level graphics APIs are generally located close to graphic displays and are used by the high level graphics API to provide and support the UI.

DEVICE IDENTIFICATION PROCESSES OF THE PRESENT INVENTION

FIG. **8** illustrates a portion **305*b*** of the HAVI communication architecture and illustrates an application layer **220**, a device control module **230** for a particular device **20**, the CMM **250**, the link driver layer **320**, an electronic device **20** coupled to the IEEE 1394 communication bus **30** and the speed map **515** and topology map **520** data structures. The location of the CMM **250** and the link driver layer **320** in the overall architecture of HAVI is shown in interface **305*a*** of FIG. **4**.

As described above, the present invention allows high level software programs (e.g., the application layer **220**) to reference devices using a persistent identification scheme. Namely, multi-bit persistent Global Unique Identifier numbers (GUIDs) are assigned to each device and contain a vendor portion and a chip series portion. In one embodiment, a 64-bit number is used as the GUID. FIG. **8** illustrates an exemplary communication path whereby the application layer **220** issues a command to cause device **20** to perform some predefined action (e.g., to start "playing" some media). The initial command from the application layer **220** includes an abstract identifier (e.g., "VCR1") of the DCM **230** object and also the action to be done (e.g., "play").

The DCM **230** of FIG. **8** converts the abstract object name into a GUID that matches device **20** and issues another command including the GUID and the action, "play," to the CMM **250**. The CMM **250** contains a GUID list **510** in accordance with the present invention. The GUID list **510** is a listing of GUIDs in order of their associated physical identifications (for each respective device). Therefore, the GUID list **510** of the present invention functions as an efficient physical identification to GUID translator and a GUID to physical identification translator. The GUID list **510** is updated by the link driver **320** each time a bus reset event is detected.

The CMM **250**, using the GUID list **510**, converts the GUID received from the DCM **230** into the corresponding physical identifier for device **20**. The CMM **250** then passes a message to the link driver layer **320** including the action command, "play," and the physical identifier corresponding to device **20**. The link driver layer **320** interfaces with the local bus **30** using the physical identifier thereby communicating the action command, "play," to the device **20**. The speed map **515** and topology map **520** are present for other aspects of the present invention described further below.

FIG. 9A, FIG. 9B and FIG. 9C illustrate exemplary speed map **515** and topology map **520** data structures. The entries of the speed map **515** and topology map **520** data structures are organized (e.g., ordered or indexed) based on the physical identifier of the devices. The mechanisms for generating the speed map **515** and the topology map **520** data structures are well known within the IEEE 1394 standard.

FIG. **9A** illustrates an exemplary topology map **520** data structure used by the present invention. This data structure

520 includes a respective entry (of entries *522a*–*522n*) for each of the n devices coupled to the local bus 30. The first field 524 of each entry corresponds to the device's physical identifier as assigned by the local bus layer 330. As shown in FIG. 9A, the entries *522a*–*522n* are ordered according to their physical identifier, e.g., physical_ID0, physical_ID1, . . . , physical_IDn. The following fields 526, 528, 530 for each entry specify port information for each physical port of the device. The port information indicates the physical connection of that port to another port. Using this port information, one can determine the manner in which the n devices of the local bus 30 are physically coupled together in the HAVI network. Topology map 520 data structures of the format shown in FIG. 9A are well known in the art and are re-generated upon each local bus reset.

FIG. 9B illustrates an exemplary one-dimensional speed map data structure *515a* used by the present invention. This data structure *515a* includes a respective entry (of entries *531a*–*531n*) for each of the n devices coupled to the local bus 30. The first field 532 of each entry corresponds to the device's physical identifier as assigned by the local bus layer 330. As shown in FIG. 9B, the entries *531a*–*531n* are ordered according to their physical identifier, e.g., physical_ID0, physical_ID1, . . . , physical_IDn. Field 534 indicates information pertinent to the communication protocol (including communicate rate) available for the device. Speed map *515a* data structures of the format shown in FIG. 9B are well known in the art and are re-generated upon each local bus reset.

FIG. 9C illustrates a two dimensional speed map *515b* data structure used by the present invention. Speed map *515b* is a matrix where each matrix entry 546 conveys information regarding the communication speed between referenced devices (e.g., devices referenced by phy_id n and phy_id 2). The speed map *515b* has a first index 542 and a second index 544, both being ordered according to their physical identifiers. Speed map *515b* data structures of the format shown in FIG. 9C are well known in the art and are re-generated upon each local bus reset.

FIG. 10A illustrates an exemplary HAVI network *560a* having five devices coupled to the local bus. Device 562 has a physical identifier (0) and a GUID(Y); device 564 has a physical identifier (1) and a GUID(Z); device 566 has a physical identifier (2) and a GUID(I); device 568 has a physical identifier (3) and a GUID(J) and device 570 has a physical identifier (4) and a GUID(X). Each of the GUID values is unique and persistent. The physical identifier values for the devices of FIG. 10A can become reassigned after a local bus reset.

FIG. 11A illustrates the resulting GUID list *510a* compiled by the present invention based on the exemplary HAVI network *560a* of FIG. 10A. The GUID list *510a* contains five entries, one for each device of the HAVI network. As shown, the GUID values in the GUID list *510a* are ordered, e.g., GUID(Y); GUID(Z); GUID(I); GUID(J); and GUID (X), according to the corresponding physical identification values, 0, 1, 2, 3 and 4, respectively. In one embodiment, GUID list *510a* is compiled by the link driver 320 and stored in computer readable memory.

FIG. 10B illustrates another exemplary HAVI network *560b* having six devices coupled to the local bus. Network *560b* is similar to network *560a* except device 566 is removed and devices 572 and 574 are added. In network *560b*, device 572 has a physical identifier (1) and a GUID (U) and device 574 has a physical identifier (5) and a GUID(N). Some of the physical identifier values between FIG. 10A and FIG. 10B have been reassigned due to a local bus reset. The reassigned physical identifiers for the remaining devices are as follows: device 562 has physical identifier (0), device 564 has physical identifier (3), device 568 has physical identifier (4) and device 570 has physical identifier (2). Each of the GUID values is unique and persistent for all devices.

FIG. 11B illustrates the resulting GUID list *510b* compiled by the present invention based on the exemplary HAVI network *560b* of FIG. 10B. The GUID list *510b* contains six entries, one for each device of the HAVI network *560b*. As shown, the GUID values in the GUID list *510b* are ordered, e.g., GUID(Y); GUID(U); GUID(X); GUID(Z); GUID(J) and GUID(N), according to the corresponding physical identification values, 0, 1, 2, 3, 4 and 5, respectively. In one embodiment, the GUID list *510b* is compiled by the link driver 320 and stored in computer readable memory.

FIG. 12A illustrates a GUID status table 630 generated by the CMM 250 of the present invention each time a local bus reset is generated. The GUID status table 630 includes entries *632a*–*632g* regarding (1) which devices remain connected to the local bus 30 since the last GUID list 510 was compiled, e.g., after a local bus reset, (2) which devices have been just removed and (3) which devices have been just added since the last GUID list 510 was compiled. A first field 632 represents the GUID value of the device and an associated field 634 represents the current status of that device. FIG. 12A represents an exemplary table 630 generated by CMM 250 as a result of the exemplary configurations of FIG. 10A and FIG. 10B. As shown in table 630, the devices having GUID(Y), GUID(X), GUID(Z), and GUID(J) remain after the local bus reset. However, the devices having GUID(U), and GUID(N) have been added after the local bus reset and the device having GUID(I) was removed after the local bus reset. In one embodiment, the GUID table 630 is compiled by the CMM 250 and stored in computer readable memory.

FIG. 12B illustrates a subset status table 650 generated by the CMM 250 which contains entries *652a*–*652b* indicating only which devices have been added to the local bus 30 since the last GUID list 510 was compiled. Likewise, FIG. 12C illustrates a subset status table 660 generated by the CMM 250 which contains entries (*662a*) indicating only which devices have been removed from the local bus 30 since the last GUID list 510 was compiled. In one embodiment, the GUID tables 650–660 are compiled by the CMM 250 and stored in computer readable memory. Device status tables 630, 650 and 660 are forwarded to objects within the HAVI architecture that establish callback handlers within the CMM 250 to receive this information.

FIG. 13 illustrates a flow diagram 700 of the steps performed by the link driver 320 of the present invention for compiling a current GUID table 510 in response to a local bus reset. After a local bus reset, physical identifiers of devices are reassigned by the local bus 30. At step 710, if a local bus reset is reported to the link driver 320, then step 715 is entered. At step 715, current GUID list is erased and the link driver 320 determines the number of devices, n, that are currently coupled to the local bus 30. A number of well known mechanisms can be used to implement step 715. At step 720, a counter is set to zero. At step 725, the link layer 320 issues a request to the device coupled to the local bus 30 and having the physical identifier corresponding to the counter value. The request is for the GUID value of this device. At this point, the link layer 320 is unaware which device this is.

At step 730, the link layer 320 receives the return GUID value from the device with the physical identifier of the

counter value and places this GUID value into a next entry of the current GUID list **510** (the entry position also corresponds to the counter value). At step **735**, the counter is incremented by one. At step **740**, a check is made if the counter is equal to n (the maximum number of devices on the local bus). If not, then step **725** is entered again to continue compiling the current GUID list. If so, then at step **745**, the link layer **320** forwards the CMM **250** a copy of the current GUID list **510**. It is appreciated that physical identifiers are assigned by the local bus **30** from 0 to (n−1) values. After process **700**, a current GUID list **510** like those shown in FIG. **11A** and FIG. **11B** is compiled.

FIG. **14** illustrates a flow diagram **800** of the steps performed by the Link layer **320** and the CMM **250** of the present invention for compiling the device status tables **630**, **650**, **660** in response to a local bus reset. As discussed above, process **700** generates a current GUID list **510** in response to a local bus reset and passes the current list to the CMM **250**. At step **820**, the current GUID list is received and the last GUID list received by the CMM **250** is retained and marked as a "previous" GUID list. At step **825**, the CMM **250** compares the entries of the previous GUID list and the current GUID list **510** and generates GUID table **650** based on the new GUID values that are present in the current GUID list **510** but not in the previous GUID list. Any object within the HAVI architecture that previously established a callback handler with the Event Manager **214** for the information of table **650** is sent a message including the status information of table **650**.

At step **830**, the CMM **250** compares the entries of the previous GUID list and the current GUID list **510** and generates GUID table **660** based on any GUID values that are no longer present in the current GUID list **510** but were present in the previous GUID list. Any object within the HAVI architecture that previously established a callback handler with the Event Manager **214** for the information of table **660** is sent a message including the status information of table **660**. Optionally, at step **830**, the information from tables **660** and **650** can be combined with the GUID values that are common to both the previous and current GUID lists thereby generating table **630**. At step **835**, the CMM **250** erases the previous GUID list and uses only the current GUID list **510**. By performing process **800**, the CMM **250** and the link layer **320** of the present invention eliminate the burden from higher level application programs of tracking which devices are on the local bus while providing the high level application programs with device status information.

FIG. **15A** illustrates a flow diagram **850** of steps performed by the present invention for translating GUID values into physical identifications using the GUID list **510** passed to the CMM **250**. At step **855**, an application program issues a command to control a device(x) and this command is received by the DCM of the device(x). At step **860**, the DCM issues a command to the CMM **250** including device(x)'s GUID value which is known to the DCM. At step **865**, the CMM **250** determines the index (e.g., entry order) of the device(x)'s GUID within the GUID list **510**. This value is the physical identifier of device(x). At step **870**, the CMM **250** issues a command to the link driver layer **320** including device(x)'s physical identifier and an action command. The link driver layer **320** then commands device(x) using the received physical identifier. Using process **850**, high level applications do not need to be aware of a device's physical identifier but can use the constant GUID value.

FIG. **15B** illustrates a flow diagram **900a** of steps performed by the present invention for accessing speed map information with respect to two devices given their GUIDs. At step **910**, an application layer issues a command to determine the communication speed supported between device(x) and device(y). The request includes the GUID values for device(x) and device(y). At step **915**, the CMM **250** receives this command and the GUID values. The CMM **250** translates the GUID values into two physical identifiers based on the index of the GUIDs within the GUID list **510**. At step **920**, the CMM **250** then accesses (e.g., indexes) a two dimensional speed map data structure **515b** with the obtained two physical identifiers. The speed map thereby returns the proper speed information. At step **925**, the CMM **250** forwards the determined speed information to the requesting application layer.

FIG. **15C** illustrates a flow diagram **950a** of steps performed by the present invention for accessing topology map information with respect to a device given its GUID value. At step **955**, an application layer issues a command to determine the topology of a device(y). The request includes the GUID value for device(y). At step **960**, the CMM **250** receives this command and the GUID value. The CMM **250** translates the GUID value into a physical identifier based on the index of the GUID within the GUID list **510**. At step **965**, the CMM **250** then accesses (e.g., indexes) a topology data structure **520** with the obtained physical identifier. The topology map thereby returns the proper topology information for device(y). At step **970**, the CMM **250** forwards the determined topology information to the requesting application layer.

FIG. **15D** illustrates a flow diagram **900b** of steps performed by the present invention for accessing speed map information with respect to two devices given their GUIDs. The difference between process **900b** and **900a** (FIG. **15A**) is that the application program determines the speed information, not the CMM **250**. At step **980**, an application layer issues a command to determine the communication speed between device(x) and device(y). At step **982**, the CMM **250** receives this command and forwards the current GUID list **510** as well as a copy of the speed map **515b** (or pointers to these data structures) to the application program. At step **984**, the application program translates the GUID values of the two devices into two physical identifiers based on the index of the GUIDs within the GUID list **510**. The application program then accesses (e.g., indexes) the two dimensional speed map data structure **515b** with the obtained two physical identifiers. The speed map thereby returns the proper speed information.

FIG. **15E** illustrates a flow diagram **950b** of steps performed by the present invention for accessing topology information with respect to a device given its GUID. The difference between process **950b** and **950a** (FIG. **15B**) is that the application program determines the topology information, not the CMM **250**. At step **990**, an application layer issues a command to determine the topology information of device(y). At step **992**, the CMM **250** receives this command and forwards the current GUID list **510** as well as a copy of the topology map **520** (or a pointer to this data structure) to the application program. At step **994**, the application program translates the GUID value of the device (y) into a physical identifier based on the index of the GUID within the GUID list **510**. The application program then accesses (e.g., indexes) the topology data structure **520** with the obtained physical identifier. The topology map thereby returns the proper topology information.

It is appreciated that by providing the GUID list **510**, the present invention allows high level applications to utilize persistent device identifiers (GUIDs) to references devices. Upon a bus reset, the high level devices are not burdened

with the reassignments of physical identifiers that are done at the local bus level. The GUID list **510** is particularly efficient by organizing the ordering of GUIDs to conform to the physical identifier value. This reduces memory required to store the GUID list **510** and creates an efficient index mechanism.

The preferred embodiments of the present invention, a method and system for providing efficient device identification using a GUID list within a consumer electronics based audio/video network, are thus described. While the present invention has been described in particular embodiments, it should be appreciated that the present invention should not be construed as limited by such embodiments, but rather construed according to the below claims.

What is claimed is:

1. A method of providing device identification to access information within a network, said method comprising the steps of:

a) constructing a list of global unique identifiers (GUIDs) wherein each GUID identifies a unique device of said network and wherein said constructing orders GUIDs within said list of GUIDs according to their associated physical identifier values, wherein said physical identifier values are assigned by a local bus of said network;

b) receiving a request originating from an application program to determine a communication speed value corresponding to a device of said network wherein said request includes a first GUID corresponding to said device;

c) using said list of GUIDs to determine a first index value corresponding to a position of said first GUID within said list of GUIDs; and

d) referencing a speed map data structure with said first index value to obtain said communication speed value, wherein said speed map data structure is organized based on physical identifiers.

2. A method as described in claim **1** further comprising the step of e) communicating said communication speed value to said application program and wherein said network is a network of the home audio/visual initiative (HAVI) architecture and wherein said local bus is of the IEEE 1394 standard.

3. A method as described in claim **1** further comprising the steps of:

f) in response to a local bus reset, renumbering physical identifiers of devices of said network that are coupled to said local bus;

g) rebuilding said speed map data structure based on renumbered physical identifiers of step f); and

h) constructing a new list of GUIDs based on said renumbered physical identifiers of step f).

4. A method as described in claim **1** wherein each GUID is a persistent multi-bit value comprising a vendor code portion and a chip series code portion.

5. A method as described in claim **1** wherein said step a) is performed in response to a local bus reset and comprises the steps of:

a1) receiving from said local bus an indication of the number of devices of said local bus;

a2) starting with physical identifier zero and incrementing to a last physical identifier corresponding to said number of devices of said local bus, forwarding requests to said devices of said local bus individually requesting their GUIDs;

a3) recording into said list of GUIDs any GUIDs returned from said devices of said local bus, said GUIDs recorded into positions said list of GUIDs based on their corresponding physical identifier values; and

a4) storing said list of GUIDs into a computer readable memory unit.

6. A method of providing device identification to access information within a network, said method comprising the steps of:

a) constructing a list of global unique identifiers (GUIDs) wherein each GUID identifies a unique device of said network and wherein said constructing orders GUIDs within said list of GUIDs according to their associated physical identifier values, wherein said physical identifier values are assigned by a local bus of said network;

b) receiving a request originating from an application program to determine topology information corresponding to a device of said network wherein said request includes a first GUID corresponding to said device;

c) using said list of GUIDs to determine a first index value corresponding to a position of said first GUID within said list of GUIDs; and

d) referencing a topology map data structure with said first index value to obtain said topology information, wherein said topology map data structure is organized based on physical identifiers.

7. A method as described in claim **6** further comprising the step of e) communicating said topology information to said application program and wherein said network is a network of the home audio/visual initiative (HAVI) architecture and wherein said local bus is of the IEEE 1394 standard.

8. A method as described in claim **6** further comprising the steps of:

f) in response to a local bus reset, renumbering physical identifiers of devices of said network that are coupled to said local bus;

g) rebuilding said topology map data structure based on renumbered physical identifiers of step f); and

h) constructing a new list of GUIDs based on said renumbered physical identifiers of step f).

9. A method as described in claim **6** wherein each GUID is a persistent multi-bit value comprising a vendor code portion and a chip series code portion.

10. A method as described in claim **6** wherein said step a) is performed in response to a local bus reset and comprises the steps of:

a1) receiving from said local bus an indication of the number of devices of said local bus;

a2) starting with physical identifier zero and incrementing to a last physical identifier corresponding to said number of devices of said local bus, forwarding requests to said devices of said local bus individually requesting their GUIDs;

a3) recording into said list of GUIDs any GUIDs returned from said devices of said local bus, said GUIDs recorded into positions said list of GUIDs based on their corresponding physical identifier values; and

a4) storing said list of GUIDs into a computer readable memory unit.

11. A method of providing device identification within a network, said method comprising the steps of:

a) constructing a list of global unique identifiers (GUIDs) wherein each GUID is persistent and identifies a unique device of said network and wherein said constructing

orders GUIDs within said list of GUIDs according to their associated physical identifier values, wherein said physical identifier values are assigned by a local bus of said network;

b) receiving a request originating from an application program to communicate with a device of said network wherein said request includes an identifier corresponding to said device;

c) translating said identifier into a first GUID corresponding to said device;

d) using said list of GUIDs to determine a first index value corresponding to a position of said first GUID within said list of GUIDs wherein said first index value is a first physical identifier; and

e) issuing a communication to said device over said local bus using said first physical identifier, wherein said communication corresponds to said application program request.

12. A method as described in claim 11 wherein said network is a network of the home audio/visual initiative (HAVI) architecture and wherein said local bus is of the IEEE 1394 standard.

13. A method as described in claim 11 further comprising the steps of:

f) in response to a local bus reset, renumbering physical identifiers of devices of said network that are coupled to said local bus; and

g) constructing a new list of GUIDs based on said renumbered physical identifiers of step f).

14. A method as described in claim 11 wherein each GUID is a multi-bit value comprising a vendor code portion and a chip series code portion.

15. A method as described in claim 11 wherein said step a) is performed in response to a local bus reset and comprises the steps of:

a1) receiving from said local bus an indication of the number of devices of said local bus;

a2) starting with physical identifier zero and incrementing to a last physical identifier corresponding to said number of devices of said local bus, forwarding requests to said devices of said local bus individually requesting their GUIDs;

a3) recording into said list of GUIDs any GUIDs returned from said devices of said local bus, said GUIDs recorded into positions said list of GUIDs based on their corresponding physical identifier values; and

a4) storing said list of GUIDs into a computer readable memory unit.

16. An electronic system having a processor, an internal bus and a computer readable memory unit, said system coupled to an IEEE 1394 local bus, wherein said memory unit having instructions stored therein that implement a method of providing device identification to access information within a network, said method comprising the steps of:

a) constructing a list of global unique identifiers (GUIDs) wherein each GUID is persistent and identifies a unique device of said network and wherein said constructing orders GUIDs within said list of GUIDs according to their associated physical identifier values, wherein said physical identifier values are assigned by a local bus of said network;

b) receiving a request originating from an application program to determine information corresponding to a device of said network wherein said request includes a first GUID corresponding to said device;

c) using said list of GUIDs to determine a first index value corresponding to a position of said first GUID within said list of GUIDs; and

d) referencing a map data structure with said first index value to obtain said information wherein said map data structure is organized based on physical identifiers.

17. An electronic system as described in claim 16 wherein said method further comprises the step of e) communicating said information to said application program and wherein said network is a network of the home audio/visual initiative (HAVI) architecture and wherein said local bus is of the IEEE 1394 standard.

18. An electronic system as described in claim 16 wherein said method further comprises the steps of:

f) in response to a local bus reset, renumbering physical identifiers of devices of said network that are coupled to said local bus;

g) rebuilding said map data structure based on renumbered physical identifiers of step f); and

h) constructing a new list of GUIDs based on said renumbered physical identifiers of step f).

19. An electronic system as described in claim 16 wherein each GUID is a multi-bit value comprising a vendor code portion and a chip series code portion.

20. An electronic system as described in claim 16 wherein said step a) of said method is performed in response to a local bus reset and comprises the steps of:

a1) receiving from said local bus an indication of the number of devices of said local bus;

a2) starting with physical identifier zero and incrementing to a last physical identifier corresponding to said number of devices of said local bus, forwarding requests to said devices of said local bus individually requesting their GUIDs;

a3) recording into said list of GUIDs any GUIDs returned from said devices of said local bus, said GUIDs recorded into positions said list of GUIDs based on their corresponding physical identifier values; and

a4) storing said list of GUIDs into said computer readable memory unit.

21. An apparatus for providing device identification to access information within a network, said apparatus comprising:

a) means for constructing a list of global unique identifiers (GUIDs), wherein each GUID is persistent and identifies a unique device of said network, by ordering GUIDs within said list of GUIDs according to their associated physical identifier values, wherein said physical identifier values are assigned by a local bus of said network;

b) means for receiving a request originating from an application program to determine information corresponding to a device of said network wherein said request includes a first GUID corresponding to said device;

c) means for using said list of GUIDs to determine a first index value corresponding to a position of said first GUID within said list of GUIDs; and

d) means for referencing a map data structure with said first index value to obtain said information wherein said map data structure is organized based on physical identifiers.

22. An apparatus as described in claim 21 wherein said means for constructing is responsive to a local bus reset and comprises:

a1) means for receiving from said local bus an indication of the number of devices of said local bus;

a2) means for starting with physical identifier zero and incrementing to a last physical identifier corresponding to said number of devices of said local bus, forwarding

requests to said devices of said local bus individually requesting their GUIDs;

a3) means for recording into said list of GUIDs any GUIDs returned from said devices of said local bus, said GUIDs recorded into positions said list of GUIDs based on their corresponding physical identifier values; and

a4) means for storing said list of GUIDs into said computer readable memory unit.

* * * * *